NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DEPLOYABLE COMMAND AND CONTROL SYSTEM FOR OVER THE HORIZON SMALL BOAT OPERATIONS

by

William D. Seegar, Jr.

September 2006

Thesis Advisor:	Craig Martell
Second Reader:	Gurminder Singh

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2006	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Deployable Command and Control System for Over the Horizon Small Boat Operations

5. FUNDING NUMBERS

6. AUTHOR(S) William D. Seegar, Jr

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT	**12b. DISTRIBUTION CODE**
Approved for public release; distribution is unlimited	A

13. ABSTRACT

The Deployable Navigation System (DeNS) is a prototype system designed to facilitate Command and Control during over the horizon small boat operations. It is designed to allow small boats to deploy from their host ships with a Bluetooth GPS (Global Positioning System) receiver and PDA (Personal Digital Assistant) running the appropriate software which provides a real time navigational picture in terms of position and relation to a predetermined track. This same data is shipped immediately back to the control ship via a wireless network and displayed on a laptop computer to allow the mission commander to monitor the small boat's progress and position, also in real time. The small boat's relation to the track is compared on every received fix and appropriate indicators are displayed to inform both users if a predetermined distance from track (track tolerance) has been exceeded. It utilizes jpg formatted maps that are derived directly from the Digital Nautical Chart (DNC) library overlaid with track information. Positions received from the GPS are converted to pixel coordinates that correspond to their original positions on the jpg chart and plotted, providing an electronic display that is very similar in appearance to the traditional plot maintained on paper charts.

14. SUBJECT TERMS Command and Control; Digital Nautical Charts; DNC; Geospatial Information System; GIS; Global Positioning System; GPS; Small Boat Operations; Over the Horizon	**15. NUMBER OF PAGES** 203
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

DEPLOYABLE COMMAND AND CONTROL SYSTEM FOR OVER THE HORIZON SMALL BOAT OPERATIONS

William D. Seegar, Jr.
Lieutenant, United States Navy
B.S., University of North Florida, 1999

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2006

Author: William D. Seegar, Jr.

Approved by: Craig Martell
Thesis Advisor

Gurminder Singh
Co-Advisor

Peter Denning
Dean, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Deployable Navigation System (DeNS) is a prototype system designed to facilitate Command and Control during over the horizon small boat operations. It is designed to allow small boats to deploy from their host ships with a Bluetooth GPS (Global Positioning System) receiver and PDA (Personal Digital Assistant) running the appropriate software which provides a real time navigational picture in terms of position and relation to a predetermined track. This same data is shipped immediately back to the control ship via a wireless network and displayed on a laptop computer to allow the mission commander to monitor the small boat's progress and position, also in real time. The small boat's relation to the track is compared on every received fix and appropriate indicators are displayed to inform both users if a predetermined distance from track (track tolerance) has been exceeded. It utilizes jpg formatted maps that are derived directly from the Digital Nautical Chart (DNC) library overlaid with track information. Positions received from the GPS are converted to pixel coordinates that correspond to their original positions on the jpg chart and plotted, providing an electronic display that is very similar in appearance to the traditional plot maintained on paper charts.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS, ACRONYMS, SYMBOLS

C2 – Command and Control
CO – Commanding Officer
COP – Common Operational Picture
COTS – Commercial of the Shelf
DeNS – Deployable Navigation System/Desktop DeNS server application
DHTML – Dynamic Hypertext Markup Language
DoD – Department of Defense
DNC – Digital Nautical Chart (Digital Library)
ET – Electronic Technician
GEOREF – World Geographic Reference System
GIS – Geographic Information System
GPS – Global Positioning System
IAW – In Accordance With
IDE – Integrated Development Environment
IEEE – Institute of Electrical and Electronic Engineers
LAN – Local Area Network
MAN – Metropolitan Area Network
MIO – Maritime Interdiction Operations
MobileDeNS – PDA DeNS client application
MSIE – Microsoft Internet Explorer
NM – Nautical Miles (2000 yards)
NMEA – National Maritime Electronics Association (GPS standard)
NPS – Naval Postgraduate School
OTH – Over the Horizon
PAN – Personal Area Network
PC – Personal Computer
PDA – Personal Digital Assistant
QVGA – Quarter VGA display (320 x 240)
RHIB – Rigid Hull Inflatable Boat
SAR – Search and Rescue
SDK – Software Development Kit
TCP/IP – Transmission Control Protocol/Internet Protocol
UAV – Unmanned Arial Vehicle
USN – United States Navy
VGA – Video Graphics Array (640 x 480)
VPF – Vector Product Format
WAN – Wide Area Network

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my thesis advisors, Professors Craig Martell and Gurminder Singh, for the immense amount of help and guidance that they provided in throughout the thesis process. Without their academic support, insight, and clarity of focus this project would not have been possible.

I would also like to thank the Information Professional community for allowing me to attend NPS and pursue work in the area of Computer Science. In addition, I would like to recognize all of my prior commands for providing me with the background and insight necessary to identify this body of work and develop a viable solution to the problems of navigation and Command and Control involved with small boat operations.

Finally, and most importantly, I would like to thank my family: my parents who have always provided me with love, encouragement, and support for all my endeavors, and my wife, Jenny, and children, David and Audrey, without whose patience, sacrifice, and support I could not be successful. Thank you all for everything you have given me.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. OVERVIEW

Upon US Naval Surface Vessels, small boats are an invaluable asset. They function in a multitude of capacities, including operation as lead lifeboats, man overboard recovery, certain mooring evolutions, Search and Rescue (SAR) operations, Maritime Interdiction Operations (MIO) and other boarding evolutions, just to name a few. The importance of the small boat to shipboard life is evidenced by its inclusion on the list of daily reports submitted to the ship's Commanding Officer (CO) for review; a report that details the status and condition of the boat and its assigned crew each day.

Typical uses of the small boat by naval vessels involve those operations which can be accomplished while maintaining visual contact between the boat and its parent vessel. Command and Control (C2) and other communication are typically provided by standard Very High Frequency (VHF) radio or other similar means. These communications are sent in the clear (i.e., not encrypted) and are subject to easy interception due to the widespread use VHF maritime radio in all seagoing vessels. Rules of the Road mandate their use on these vessels, and by placing the radio in scan mode, multiple frequencies can easily be monitored.

Navigation on the small boat is carried out by seaman's eye, steering by instruction passed through the VHF radio, and/or the use of a magnetic compass. Paper charts can provide some assistance to the boat crew through the use of estimation techniques such as dead-reckoning (DR) to help identify aids to navigation and or pass general area coordinates, but they are cumbersome to use and highly subject to weather conditions, sea spray, high winds due to the boats movement through the water, etc.

While some naval vessels have developed "home grown" solutions to provide a better capability for C2 and general communication with the small boat, no standard solution is available—none that can be easily implemented, is portable across platforms, is both accurate and extendable to ranges that would

facilitate over the horizon (OTH) operations, and that utilizes naval standards by leveraging current technology in terms of devices, architectures, and systems. The Deployable Navigation System (DeNS) prototype system is a first attempt to do just those things.

B. PROBLEM SPACE

With the emergence and increasing popularity of mobile device technology and the increasing acceptance of wireless network standards and architecture, the US Navy should look for ways to leverage these advancements in order to improve its everyday operations. The DeNS prototype system concentrates on a way of achieving this goal in regards to C2 of small boat OTH operations. By utilizing mobile devices—such as the Personal Data Assistant (PDA) and laptop computer, as well network architectures as defined by the Institute of Electrical and Electronic Engineers (IEEE) standards 802.11 (WiFi), 802.16 (WiMax), and 802.15.1 (Bluetooth) in conjunction with the Global Positioning System (GPS) and Digital Nautical Chart library (DNC)—the DeNS is able to provide a viable, easily obtainable, and highly deployable C2 system for use in this arena.

To date, the DNC provides the only digital cartography product that has been authorized for use in the navigation of US Naval vessels by the Chief Navigator of the Navy. While its use in shipboard systems is currently being implemented, there are no efforts being made to port this powerful geospatial database onto a PDA. The DeNS prototype system makes in-roads into this endeavor by allowing the user to create accurately scaled jpg formatted digital images, similar in appearance to paper charts, which are derived directly from the DNC. These images can then be transferred to the PDA for use with the DeNS prototype system. While this is not the optimal application of DNC data on the PDA (further work as will be discussed in Chapter V will explore full DNC implementation on the PDA device,) it does provide the first step towards its implementation.

While GPS systems currently exist and are commercially available that can provide the small boat with accurate positional information, they neither implement the DNC, nor allow for communication and data relay between the small boat and its parent vessel. These aspects of a deployable system are crucial to establishing the C2 functions necessary to enable a CO to comfortably authorize extended OTH operations. Further discussion of this aspect of these systems, as well as descriptions of “home grown” systems will take place in Chapter II, Section E.

Of primary importance in the development of the DeNS prototype system was the ability to provide a means of two way communication for both GPS positional data, as well as other supplemental information such as the issuance of orders and compliance with those orders, deviations from a designated track determined prior to mission start-time, and the possibility to transmit other data packets such as photographs or scanned documents. While not all of these were implemented in the prototype due to time constraints, those which were omitted can be easily added in future iterations of the system due to the built in flexibility provided by the design of the server/client model that was implemented. By creating new packet types, the client and server applications can easily be made to handle new types of data in the appropriate manner.

C. OBJECTIVE

DeNS is a prototype system designed to facilitate C2 during OTH small boat operations. It is designed to allow small boats to deploy from their host ships with a GPS receiver and PDA running the appropriate software which provides a real time navigational picture in terms of position and relation to a predetermined track. This same data is shipped immediately back to the control ship and displayed to allow the mission commander to monitor the small boat’s progress and position, also in real time. The small boat’s relation to the track is compared on every received fix and appropriate indicators are displayed to inform both users (in the small boat and onboard the control ship) if a predetermined distance from track (track tolerance) has been exceeded.

As described above, the objective of this project was to build a proof of concept system which implemented the DNC, GPS, and a wireless network with ranges sufficient to facilitate C2 and a common operation picture (COP) that is shared between the small boat and its parent vessel. Of primary importance in this phase of the project was the implementation of the DNC data on a PDA platform, and proper rendering and plotting of positional data. Both of these objectives were met, as will be demonstrated in the remainder of this paper.

Our main concern during the inception of the DeNS Prototype system thesis was: Can the DNC be exported to a windows based pocket personal computer (PC) in manageable pieces to allow for OTH small boat operations and two way data update in a manner that doesn't impede operator mission success? The answer to this question was a resounding "yes." As with most questions of this overarching nature, many subsequent, more pointed questions resulted. These included such items as process automation, ease of use, the best user interface and network architectures to implement, etc. The remainder of this paper will address these concerns, with results summarized in Chapter V.

D. SCOPE

The DeNS prototype system is a proof of concept system as described earlier in this chapter. Specifically, development included the implementation of applications on both the PDA and the control ship to facilitate the display of jpg images derived from the DNC with applicable track overlays, GPS data display, and addressed the communication of such information over a portable protocol that will work across network topologies. The proof of concept presented here focused on the use of an 802.11 wireless Local Area Network (LAN), but should be expanded to a more able network implementation in the fielded system, as discussed in greater detail in Chapters II and V. Additionally, while the focus for the proof of concept is the sharing of a navigational track and related information, we will present recommendations for future expansion in Chapter V.

As with any proof-of-concept constraints on the operation of the DeNS prototype system were also necessary. Most of these limitations to scope are discussed in Chapter IV, sections B.4 and C.2.d. They include network implementation on 802.11 vice 802.16, and the operational constraints in terms of latitude and longitude to the inclusive degrees of 36^0 N and 121^0 W. Further, at this point only positional data and message data transmissions are implemented, and only in one direction (small boat to control ship), though this can easily be expanded.

E. THESIS ORGANIZATION

This thesis consists of five chapters and three appendices. Chapter II will provide foundational background information on which this thesis was based, along with a brief discussion of existing related works focusing on both Commercial Off the Shelf (COTS) products, and US Navy "home grown" solutions. Readers not interested in detailed descriptions of development environments and device specifications can safely skip Chapter II, Section C. Chapter III will provide an architectural description of the various components that comprise the DeNS prototype system. Chapter IV will provide insight into the implementation and usability of the system, along with its known bugs and limitations. Finally, Chapter V will provide conclusions, recommendations for future works, and a summarization of the project.

Appendix I and II both provide a listing of the MobileDeNS and DeNS C# code, respectively, along with applicable comments. Throughout this thesis certain conventions are used when referring to the DeNS system. The DeNS prototype system is comprised primarily of two applications, one designed to run as a client on the PDA, which will be referred to as MobileDeNS, and one designed to run on a laptop or desktop computer, referred to as DeNS. For clarity, any reference to "the DeNS", "the DeNS system", or "the DeNS prototype system" refers to the DeNS system as an overarching entity comprised of all of its various components. Reference to "DeNS" simply refers to the server portion of the system that is designed to run on the laptop.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND AND RELATED WORK

A. OVERVIEW

While much work has been done in the civilian sector to develop and market handheld navigation devices to recreational boaters and professional mariners, they do not address certain requirements for the US Navy. Specifically, they use charting tools other than the DNC Library, which is the only digital media authorized for navigational use by the Navigator of the Navy (USN, GIN 002, GIN 003, GIN 004). Additionally, they do not address the need for C2 of the small boat by the boat's parent vessel, nor do they allow for a COP.

In this chapter, we will discuss the choices made in the development of the DeNS and the different device components used in the prototype and their specifications. Also covered will be the aspect of content repurposing that was required to implement the DNC on a handheld device, exactly what the DNC is and how it works. Finally, we present an overview of related work within the US Navy in the fielding of various systems intended for a similar purpose as that of the DeNS.

B. NETWORK CAPABILITIES

When considering network capabilities, it is important to note the immediate differences between the various types of networks as a whole. Networks can range in size from the small Personal Area Network (PAN) that has a range of a few meters, to the much larger Metropolitan Area Network (MAN) and Wide Area Network (WAN) that can span cities and even states, respectively (Antonello).

The PAN is implemented using technologies such as Bluetooth to create a short-range network through which Bluetooth enabled devices can communicate. This is most common in the cell phone and PDA arena, where devices utilize the Bluetooth connection for connectivity between accessories such as headsets, or for synchronization purposes between devices such as PDA's, cell phones, and

desktop computers. It is also expanding to include peripheral devices such as printers, scanners, and similar devices to allow wireless transmission of data between said devices. DeNS uses a PAN via Bluetooth to facilitate communication between the PDA running MobileDeNS and a Bluetooth enabled GPS receiver, both of which are detailed below.

The MAN is typically a collection of nodes within a metropolitan area that fall under the same corporate control; possibly a telecommunications company or an ISP. The wireless medium used most often in this area of networking is the IEEE 802.16 standard (Antonello). Table 1 presents a quick comparison of the 802.16 standard with the WLAN standard, 802.11. Speed and area of coverage, both of major concern for the DeNS implementation, is shown to be superior for 802.16 in standard installations. Particularly important to DeNS is the range of coverage. Assuming the standard visible horizon for a naval vessel is 10 Nautical Miles (NM) (this is a rule of thumb as the actual horizon distance will vary based on height of eye), it is clear that 802.11 will not achieve the ranges needed for a major goal of the DeNS, C2 for OTH operations. As shown, 802.16 can achieve ranges up to 30 miles with adequate antennae height and transmitter power. The aspect of included bands is not addressed in this paper beyond what is shown, as both are capable of utilizing unlicensed bands, which is the desired state for DeNS.

	WiFi 802.11	**WiMax 802.16a**
Speed	6-54 Mbps	70 Mbps
Band	Unlicensed (2.4 and 5 GHz)	Both (2.5-3.8 & 5.7-5.8 GHz)
Coverage	50-1500 ft	2-30 Miles

Table 1. Quick Technology Comparison after (Airespace Technologies).

1. Development Medium

Having acknowledged the requirement of 802.16, or a similarly positioned network standard, the decision was made to implement the prototype for DeNS on an 802.11 network. The primary reasons for this decision were the availability of an already existing 802.11 network, the cost and time constraints that would have been associated with planning and implementing an 802.16 network solely for the development of this prototype, and the fact that despite the differences cited, both standards are equally operable with the Transmission Control Protocol/Internet Protocol (TCP/IP) which is the basis of communication for the DeNS.

2. Implementation Medium

As discussed above, any further development of the DeNS and/or fielding of the DeNS should be done with an 802.16 implementation in order to ensure the OTH ranges are achieved. The speed advantage of 802.16, while not paramount in the prototype which is sending relatively simple, small packets across the network connections, could become a much more important factor if the system were expanded to allow for more complex traffic such as photographs, scanned documents, or streaming video.

C. ENVIRONMENT / DEVICE SPECIFICATIONS

The DeNS was developed using the equipment, software, and Integrated Development Environment (IDE) listed as follows, detailed below:

- Development PC: Toshiba Tecra M4-S435
- IDE: Microsoft Visual Studio .NET 2003 Enterprise Architect Ed.
- PDA: HP iPAQ hx4700
- GPS: HP iPAQ Navigation System
- Master Station: Toshiba Tecra M4-S435
- Third Party Software: MultiIE v3.1-d59

- Third Party Software: GPSAccess v1.0.0
- Third Party Software: ArcGIS v9.1
- Third Party Software: ArcIMS v9.1
- Network Technologies: Bluetooth, IEEE 802.11, TCP/IP

1. Development Environment Specifications

All coding requirements for the DeNS were implemented using a Toshiba Tecra M4-S435 as detailed in Table 2. DeNS and MobileDeNS code (found in Appendix I and II, respectively) were written in C# using the Microsoft Visual Studio .NET 2003 Enterprise Architect Edition IDE.

OS	Microsoft Windows XP Tablet Edition
CPU	Intel Pentium M-740
RAM	512 MB DDR2 SDRAM
Network capabilities	V.92 Modem Gigabit Ethernet Intel Pro/Wireless 2200BG (802.11b/g)

Table 2. Development PC (Toshiba Tecra M4-S435) Specification.

2. DeNS Device Specifications

The DeNS is primarily comprised of three devices: a PDA; a GPS receiver; and a laptop or desktop computer, hereafter referred to the master station. All three device specifications used in the prototype are as follows:

a. PDA

MobileDeNS runs on the PDA using a GPS receiver which is discussed below. The PDA used for implementation of the prototype system is

an HP iPAQ hx4700 as detailed in Table 3. This PDA was appealing for the prototype primarily for its full video graphics array (VGA) display capability, vs. the more common quarter video graphics display (QVGA) on competing PDA's. For actual fielding of the system, a ruggedized solution should be implemented to ensure the PDA device is protected from factors existent in the small boat environment, such as inclement weather, sea-spray, and rough treatment of the device. This could be accomplished through the use of either a ruggedized PDA, or a ruggedized case, both of which are readily available on the commercial market.

OS	Microsoft® Windows Mobile™ 2003 Second Edition
Processor	Intel PXA270 Processor 624 MHz
Memory	192 MB total memory; 128 MB Strata Flash ROM and 64 MB SDRAM
Display	4" Transflective VGA TFT color display
Interface	USB 2.0, Infrared (SIR and FIR), Bluetooth, WiFi (WLAN 802.11b)
Expansion	SD card slot, supports MMC and SDIO cards, CF Type II card slot

Table 3. PDA (HP iPAQ hx4700) Specification.

b. GPS Receiver

The GPS receiver used was the unit which comes with the HP iPAQ Navigation System bundle. There is no programmable Application Programming Interface (API), nor otherwise accessible methods to communicate with the unit as packaged, nor does HP provide any additional support for this functionality. The GPS unit uses Bluetooth technology to communicate with the PDA, however, so third party software can be purchased to allow access to the

GPS receiver and its data. The software used in this prototype was GPSAccess, which is discussed at more length below.

c. Master Station

The master station used for prototype development was the same Toshiba Tecra M4 tablet PC that was used for development. DeNS runs on this platform as a C# windows application, so it should run on any Windows platform with the .NET framework installed.

3. Third Party Software Specifications

Four pieces of third party, commercially available, software were used in the development and implementation of DeNS: MultiIE for browser functionality; GPSAccess for Bluetooth connection with, and data extraction from, the GPS receiver; ArcGIS Suite for chart creation; and ArcIMS to facilitate web sharing of charts created using ArcGIS. Further details are as follows:

a. MultiIE v3.1-d59

MultiIE is a browser plug-in built specifically for mobile devices by Southway Mobile Applications. Considering the limitations inherent in mobile devices, many software applications are stripped of features which are not considered necessary for mainstream use (Yao). This is true of Microsoft Internet Explorer (MSIE) 4.01 which is the browser included in the Microsoft Windows Mobile2003 Second edition OS. Of particular interest to the DeNS system MSIE 4.01 does not allow images (.bmp, .jpeg, etc…) browsed to be saved on the mobile device on which it is installed (the “Save picture as…” option that is accessible by right clicking an image in the browser). This is the primary means to load the chart images from the ArcIMS server, which will be detailed below. To add this functionality back into the browser, MultiIE was used, as it includes this feature as well as many others.

b. GPSAccess v1.0.0

As was mentioned earlier, GPSAccess is a third party solution created by High Point Software which comes complete with a C# library to allow customized programs to access Bluetooth enabled GPS devices. GPSAccess primarily consists of two components, the library mentioned above, and GPSLink, an application that runs on the device with which you wish to connect to the GPS receiver, in our case the PDA (Highpoint Software).

The included library integrates the .NET Compact Framework v2.0 with the Widcomm Bluetooth stack v1.6 and later. The integration of the Widcomm is of primary significance here, because it allows a seamless user interface without requiring the specification of ports or the need to choose a device from a list of all possible devices. It connects directly to the GPS receiver without prompting. Additionally, the library allows the DeNS to access data directly without the need to parse the National Maritime Electronics Association (NMEA) formatted string for all information needed, and allows direct control and monitoring of GPS receiver status (Highpoint Software).

The second component, GPSLink is the application that manages the connection to the GPS receiver for the PDA. Using this tool, not only do you connect to the GPS receiver, but you can also monitor its status, readings, and control it manually through a variety of settings (Highpoint Software).

c. ArcGIS v9.1

ArcGIS by ESRI provides the method through which the DNC can be viewed and manipulated to build the nautical charts required by the DeNS. It is a Geographic Information System (GIS) product that reads a variety of file and database formats, though we are only interested in its ability to render Vector Product Format (VPF) based files, as this is the standard format used by the Department of Defense (DoD) to create digital maps and charts (ESRI GIS an Mapping Software). It consists of several parts, but of primary use in the DeNS are the ArcCatalog and ArcMap applications, as well as the Military Analyst

extension. While details of each of these products, and their respective use in DeNS will be detailed in section D, part 2, of this chapter, a brief explanation of each product is given below.

ArcCatalog is the application included in the ArcGIS suite that organizes and manages all GIS data including maps, charts, and metadata. With ArcCatalog the user can browse geospatial data, manipulate metadata, and create, define, and export and import geospatial databases (ESRI GIS and Mapping Software). This is the starting point for building DNC charts for use with DeNS.

ArcMap is the central application included in the ArcGIS suite. It allows map analysis and editing of maps; as well as export functionality in a variety of formats, including jpg formatted images (ESRI GIS and Mapping Software). This tool is used to build and edit the charts for use with DeNS.

By applying the Military Analyst extension within both ArcCatalog and ArcMap, both products are able to read, understand and render the VPF data contained in the DNC. By default, ArcGIS does not support this format. VPF is a standard that was created specifically for DoD use in digital geospatial products (USN, Interface Standard for Vector Product Format). As such, it is not the industry standard; however, since ESRI played a large part in the creation of the VPF standard, they have developed the Military Analyst to ensure compatibility with their products (ESRI GIS and Mapping Software).

d. ArcIMS v9.1

ArcIMS is a web based GIS Server that delivers GIS data in a format that can be accessed through a network, be it the World Wide Web, or a local intranet. It is completely compatible with other ArcGIS products and can read and render GIS data created with ArcGIS or images based on said data (ESRI GIS and Mapping Software). DeNS currently uses this product as the

delivery tool for image based charts to be deployed to the PDA, though the possibility exists to extend its functionality, as will be discussed in Chapter V, section C.

D. CONTENT REPURPOSING

Most of today's online content is built to be viewed on a conventional desktop PC. While this meets the needs of a large majority of online users, it can often fall short of optimal, or sometimes even usable, parameters when accessed by users of mobile devices such as PDA's. This is due to the inherent differences between the two platforms in general (Singh, Content Repurposing).

Table 4 offers a generic comparison of the desktop platform vs. the mobile device. As can be seen, some consideration must be taken in planning to deliver content of any kind to both desktop PC's and mobile devices. This consideration is known as content repurposing. In the DeNS implementation this becomes important in the handling of the DNC chart information.

Issues	Desktop	Mobile Devices
Processor	Up to 3+ GHz	Up to ~624 MHz
Memory	Up to 2+ GB ram alone	~192 MB total memory
Power	Unlimited, continuous	Limited by battery life
Display	Up to 17+ inches, high res	Up to ~4 inches, QVGA
User Interface	Full keyboard, mouse	Stylus, button driven
Connectivity	Constant connection capable	Less robust, volatile
Bandwidth	Up to 384+ kbps	Typically lower, volatile

Table 4. Typical Desktop vs. PDA Quick Comparison.

Content repurposing as whole can be implemented in a variety of methods (Singh, Device Independence-II):

- Post-processing – Content repurposing is conducted dynamically at the time it is requested.
- Pre-processing – Content repurposing is conducted before requests are processed, and mechanisms are in place to help facilitate the correct objects being utilized.
- Server-side processing – All content repurposing is conducted at the server, freeing the client resources for other work.
- Client-side processing – All content repurposing is conducted y the client, freeing server resources for other work.
- Proxy based processing – Client requests content from a proxy which retrieves the content and repackages it as necessary for delivery. This frees resources on both the client and server side.

While each method listed above has its advantages and disadvantages, most often they are implemented in a hybrid manner. That is, one or more of the above techniques are used in conjunction to try to achieve optimal retrieval while freeing resources wherever possible. The DeNS utilizes such a hybrid scheme, implementing a pre-processing, server-side implementation for chart retrieval by the PDA. This approach provides less taxing of the more limited PDA, while requiring slightly more code in the ArcIMS server. The pre-processing portion of the DeNS content repurposing scheme refers to the fact that the charts are prepared and placed in the server before any request is ever made for them in both .axl format (for web display using ArcIMS) and .jpg format for DeNS and MobileDeNS.

1. Digital Nautical Chart Library

The DNC is a vector-based digital database library providing coverage of over 5,000 nautical charts intended to provide world wide coverage between the

latitudes of 84° N and 81° S (USA, Performance Specification: Digital Nautical Charts). It uses the VPF format, as mentioned above, which was developed in cooperation with ERSI, the manufacturer of the Arc line of GIS products, and has the distinction of being the only digital chart media authorized for navigation by the Chief Navigator of the Navy (USN, GIN 002, GIN 003, GIN 004).

The DNC itself consists of 29 CD-ROM libraries, each providing non-overlapping coverage of a specific area of the globe as shown in Figure 1. Libraries are organized using the World Geographic Reference System (GEOREF); a system that divides the surface of the earth into quadrangles, each with sides that are specific lengths of arc of latitude and longitude as appropriate, and identified by a systematic letter code to provide positive identification. Each quadrangle can be considered a tile, and further divided into smaller tiles. The DNC consists of four levels of tiles, based on the original paper chart scale and type, as shown in Table 5 (USA, Performance Specification: Digital Nautical Charts).

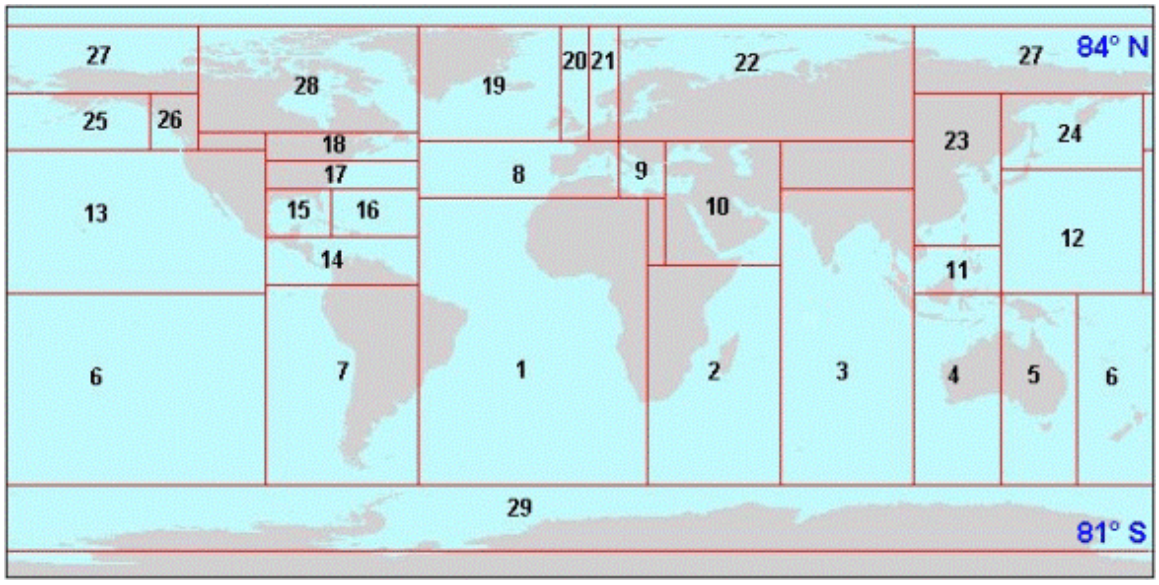

Figure 1. DNC Library Coverage from (USA, Performance Specification: Digital Nautical Charts)

Library (Chart Type)	Tile Size	Chart Scale
GENERAL	3°	< 1:500,000
COASTAL	3°	1:75,000 – 1:500,000
APPROACH	30’	1:25,000 – 1:100,000
HARBOR	15’	> 1:50,000

Table 5. DNC Tile Sizes from (USA, Performance Specification: Digital Nautical Charts).

Data stored in the DNC is divided into libraries based on the chart types as shown in Table 5 on the individual CD-ROM’s. The GOEREF identifier is used as the directory name that contains all primitives related to that particular tile, and the tiles for each chart scale band reside in different libraries allowing unique identification through their path name (i.e., DNC03\GEN03\MKNN is a general library, while DNC03\COA03\MKNN is a coastal library, each a 3° tile in the DNC03 database) (USA, Performance Specification: Digital Nautical Charts).

Considering the shear amount of data which must therefore be contained on each CD-ROM, and the corresponding small area being utilized for small boat operations, it becomes obvious that some type of content repurposing is necessary to prevent overburdening the PDA device, the limitations of which have already been discussed. As previously stated, DeNS utilizes a pre-processing, server-side repurposing scheme along with VPF to JPG conversion to properly implement the DNC based charts that it uses as a basis for navigation.

2. Creating Extents with ArcMap

An extent is simply the portion of the GIS product that is currently being used in the ArcGIS suite. While the DNC is not technically a GIS product, with the use of the Military Analyst extension discussed in section C.3.c. of this chapter, we can use the DNC in the same manner as a GIS product. Before this

can be done, however, there are some preliminary steps that must be completed (ESRI GIS Mapping and Software).

Obviously, the ArcGIS suite must be properly installed and functioning, and specifically, both the ArcCatalog and ArcMap products must be components of that install. Additionally, the ArcToolbox should be installed, and the Military Analyst extension needs to be activated in both ArcCatalog and ArcMap (ESRI GIS Mapping and Software). Once this is complete, the process of creating an extent can begin:

1. In ArcCatalog a new personal geodatabase must be created. This is accomplished by simply right clicking in the explorer window in the appropriate directory in which you wish the database to reside, and choosing the "New>Personal Geodatabase" option from the resulting menu. Name the database as desired.

2. Load DNC data into the geodatabase just created.

a. Before this step can be accomplished, it may be necessary to copy the actual database contents from the CD-ROM to your hard drive. This was necessary in the development of the DeNS prototype, though whether this was due to hardware limitations or ArcGIS inherent limitations remains unclear, even after contact with ESRI support facilities.

b. After the DNC is copied to the local machine, you can proceed to add it to the geodatabase. This is accomplished by right clicking the geodatabase in the explorer window and selecting "New>MA Catalog". This will open a dialog window in which you will specify the name of catalog to be created, the location of the data to be added, and the format of the files to be added. The location should be the top level directory copied from the DNC CD-ROM, and the file type should be VPF Catalog.

At this point there should be a working MA Catalog with the applicable DNC data added to it. We can now proceed to ArcMap to begin defining the extent we will use in the DeNS.

1. Open ArcMap and click the “Add MA data” button on the Military Analyst toolbar. By default this will reside in the bottom right-hand part of your screen as shown in Figure 2. This will open a dialog box asking for the name and type of catalog to be added. Specify the type to be “MA VPF Catalogs” and Browse to the MA catalog in the geodatabase created previously. Click Add.

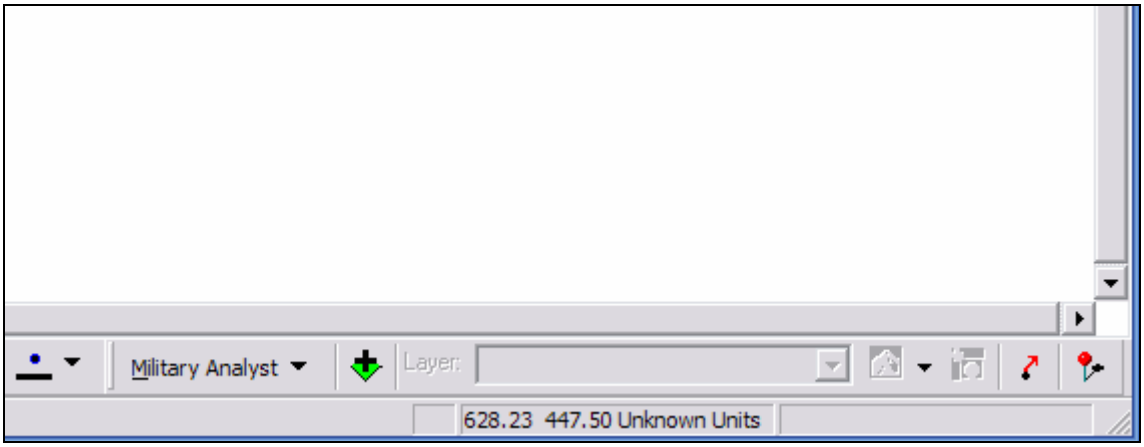

Figure 2. Add MA Data button (black plus over green tile).

2. The wireframe representing the GOEREF tiles will be displayed initially. In the explorer window, right click on the top level layer just added, and choose properties.

3. In the VPF layers tab under VPF Display Options, check “Display VPF Data for Selected VPF Products”. In the VPF Layer Selection window on the right, we can browse and choose those layers we wish to view in our extent. Click OK.

With the DNC data made available for viewing, we can now zoom to the applicable area in which small boat operations will be conducted. ArcMap allows complete customization of the chart display including color and/or patterns used to display features, turning features on or off as needed, and rearranging the drawing order of said features to ensure important items are visible and not overlooked. This is an exercise left to the discretion of the user, as it will be

somewhat operationally dependent, and the details of ArcMap use beyond what has already been given is beyond the scope of this paper.

3. Exporting Extents for use with DeNS

Once the extent that will be used has been created, it is a relatively simple task to export it. Simply choose the Export option from the file menu and it will allow you to produce a .jpg image of the extent currently being viewed. This same extent should be saved for later use with ArcIMS as will be detailed in Chapter 3. Also of note, the extreme latitudes and longitudes of the current extent should be recorded by hand as there is currently no automation involved to export/import them into either DeNS or MobileDeNS. They will, however be required for both applications to properly plot the GPS coordinates on the jpg representation of the DNC based chart. Also required will be the pixel height and width of the resulting jpg.

E. RELATED WORK

1. Overview

Related works in the area of interest, specifically network-enabled navigation systems, will be covered in this section. To illuminate various sectors of products and research, this section will provide a broad overview of COTS products available for similar purposes, United States Navy (USN) specific products intended for small boat C2, and finally research currently underway to facilitate networking at the desired ranges for OTH operations.

2. COTS Products

The commercial sector provides many options for both mounted and handheld GPS navigation at sea. Manufacturers such as Garmin, Magellan, and Cobra all provide solutions ranging in price from $100 to $1,000+ that are intended primarily for recreational boaters, or mariners operating independently. These products can be easily purchased from a local marine store or the internet,

but most of these low cost, easily available products do not provide any networking features that would allow for a COP between units, nor do they utilize the DNC, both of which are necessities for a naval vessel conducting small boat OTH operations.

The network tracking aspect of the navigation system can be obtained through the implementation of either a commercially managed navigation tracking system, or by the purchase of a much more elaborate self managed system. Both of these solutions are routinely utilized by commercial fishing fleets, merchant marines, cruise lines, etc. However, in both types of systems, price can be prohibitive, as subscription rates, equipment costs and the cost of satellite communications can reach figures which preclude this option based on the relative time it would be utilized by a naval vessel and its respective small boat. Further, the scalability of these solutions, from fleet-size implementations to a 2-3 vessel implementation at most, makes them impractical for consideration as viable solutions for individual naval units, and again, they do not utilize the DNC.

3. USN Products

In 1999, Electronic Technicians (ET's) onboard USS FLETCHER developed a system referred to as NavTrack in order to aid in conducting MIO's in the Northern Arabian Gulf. This system is the only system of its kind discovered during research for the DeNS project. It was designed for the same purpose, namely small boat C2, but was limited to a range of 5 NM, which would not allow for the small boat to be deployed OTH. Additionally, visual display was available only onboard FLETCHER, not in the small boats themselves, and even onboard FLETCHER the electronic chart representation was achieved through commercial products, rather than the DNC (Earhart).

NavTrack utilized Ross DSC 500 P radios, Furuno GPS units, and a wireless LAN known as RIBNET which was capable of transmitting digital imagery, scanned text files, sound and video at speeds up to 1.54 MB per

second at distances up to 5 NM. The Ross radios interfaced with the GPS receivers to display ownship's position and fly to points on a dot matrix display, as well as allowing for polling and display of other ships positions equipped with similar radios. In the FLETCHER implementation, each Rigid Hull Inflatable Boat (RHIB) and the FLETCHER pilot house was equipped with such a unit (Gombert).

In order to display the positions received at the pilot house, Hi-plot software developed by West Marine of Hawaii was used on a standard laptop to control the radio and plot the received polling data on an electronic chart. Hi-plot controls the Ross radios via an RS232 cable to a combiner used to multiplex radio in, radio out, and NMEA data sent to and from the laptop's serial port. Multiple RHIB data could be displayed along with ownship's positional data, but the program was designed to only work with the Ross radios and the MS Windows XP Operating System exclusively. Issues were noted when trying to install the software to any other OS, including other versions of MS Windows (Gombert).

4. Wireless Networking

As has been discussed repeatedly, the networking aspect of the DeNS is crucial. It allows for a COP between parent vessel and small boat, as well as the passing of other mission critical information such as imagery, orders, and interrogative messages. As was discussed in section B above, the IEEE 802.11 standard was used to implement the networking portion of the DeNS. While this medium models the necessary networking ability of the system well, it is not appropriate for fielding the actual DeNS due to range limitations. For this reason another medium must be implemented when fielding this system.

IEEE 802.16 has been recommended as it closely resembles the development medium to the end applications. The 802.16 implementation could be achieved in a variety of ways. One possible implementation would be through the use of antennas mounted on the highest mast of the ship and in the small

boat itself, and/or the use of small Unmanned Arial Vehicles (UAV's) to act as relay stations between the two communicating end points. In practice however, any medium which is capable of transmitting the TCP/IP packets and achieving the appropriate ranges could be used in its place. This could include satellite communications, Iridium phones, or any other medium over which the master station and PDA could be connected.

Beyond this, the question of networking in the field is beyond the scope of this thesis, though it will be touched on again in Chapter V as future work.

F. SUMMARY

In Chapter II we have focused on the background, development, and existing works related to the DeNS. Networking was shown to be a crucial aspect of the systems as it is required for not only communication between the GPS device and the PDA, but also to facilitate the sharing of that data with the master station onboard the parent vessel. Further, ranges of up to 15NM were established as the minimum requirement for such a system in order to allow OTH operations, and IEEE 802.16 was recommended as the medium of choice.

Specifications were provided for the DeNS devices as well as the environment used to create the prototype system. Third party software were also discussed in terms of their respective fit and functionality in the system, and a brief description of the DNC was provided for insight into it's operation, functionality, and the role that content repurposing plays in its implementation. Finally, a brief overview of related works was provides which focused on naval applicability and the sole existing system discovered during the research phase of this project.

Before discussing actual usage and implementation of the DeNS, we will look further into the architectural details of the system in Chapter III. This will provide diagrams and data flows to help better understand what is happening in the system as various functionalities are discussed in the Chapter IV.

III. ARCHITECTURE DESCRIPTION

A. OVERVIEW

Chapter III is intended to provide a better understanding of the specific architectural elements of the DeNS and the interaction between those various elements. In order to better illustrate the prototype system's architecture, we will first discuss the overall architecture, followed by a more in-depth look at the architectural elements in both MobileDeNS and DeNS, complete with data flow descriptions and diagrams.

B. OVERALL ARCHITECTURE

The overall prototype system architecture of the DeNS is shown in figure 3 below. As shown, there are six primary elements of the prototype system: 1) the Master Station; 2) the DNC database; 3) Web/GIS Server; 4) PDA and GPS receiver; 5) GPS constellation; and 6) the Surplusdedicate 802.11 network. Of the six elements, two will not be discussed at depth as they either exist external to the scope of the DeNS, or are being used to facilitate prototype testing only. The two elements in question are the GPS Constellation and the Surplusdedicate wireless network.

The GPS Constellation is a fully functional satellite network that is already in place and managed by government agencies. While it is an integral part of the DeNS' functionality, it is an external element and as such is beyond the scope of this project. It is shown to complete the diagram, but only the data gathered by the receiver may be manipulated by the system, therefore, no further discussion will be addressed to this aspect of the DeNS prototype.

The second element of the diagram which will not warrant further discussion is the Surplusdedicate wireless network. This is an IEEE 802.11 network (SSID Surplusdedicate) located on the Naval Postgraduate School (NPS) campus that is managed and ran by the Information Technology and Computer Services office of NPS, which operates and/or maintains all school

ADP equipment. As has been discussed at length in Chapter II, 802.16 is recommended for fielding of the DeNS system.

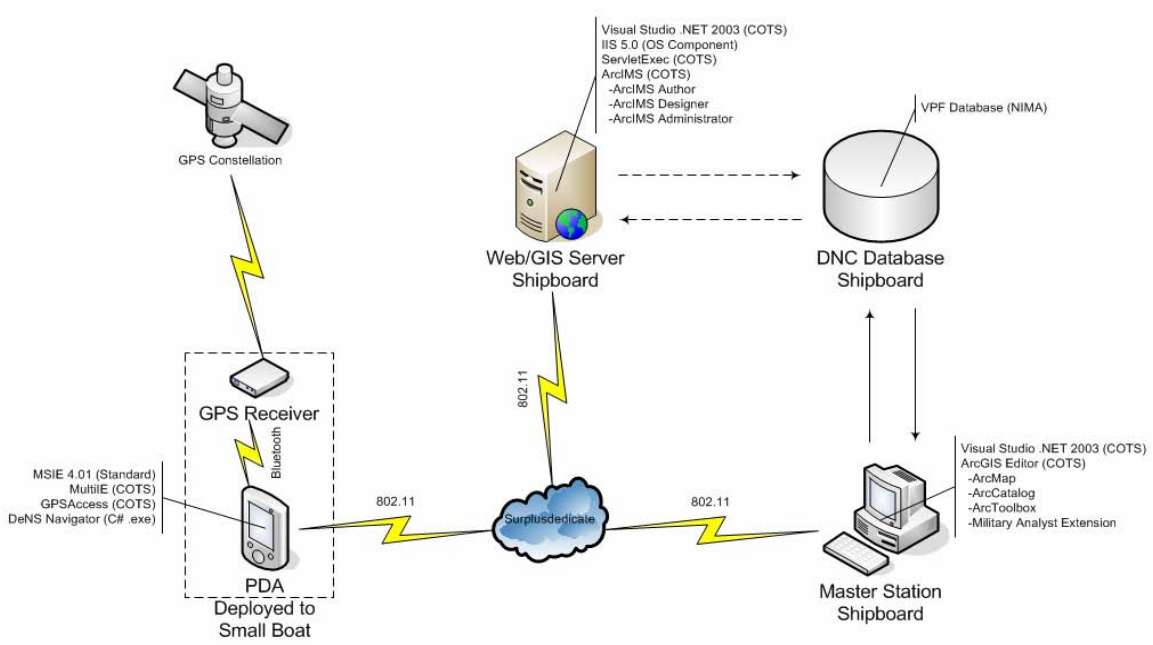

Figure 3. DeNS Architecture.

The remaining four elements can be divided into two distinct groups, MobileDeNS and DeNS, the distinction being all MobileDeNS equipment is deployable on the small boat, while all DeNS equipment is maintained onboard the parent vessel.

C. MobileDeNS

MobileDeNS is that portion of the system that is deployable to the small boat, i.e., the GPS receiver and the PDA. Figure 4, below, provides an overview of the architecture that comprises this portion of the DeNS prototype system. The blow up of the PDA shows the software interaction as well as input and output that is required or produced, respectively. Before detailing the MobileDeNS application, third party applications will be discussed.

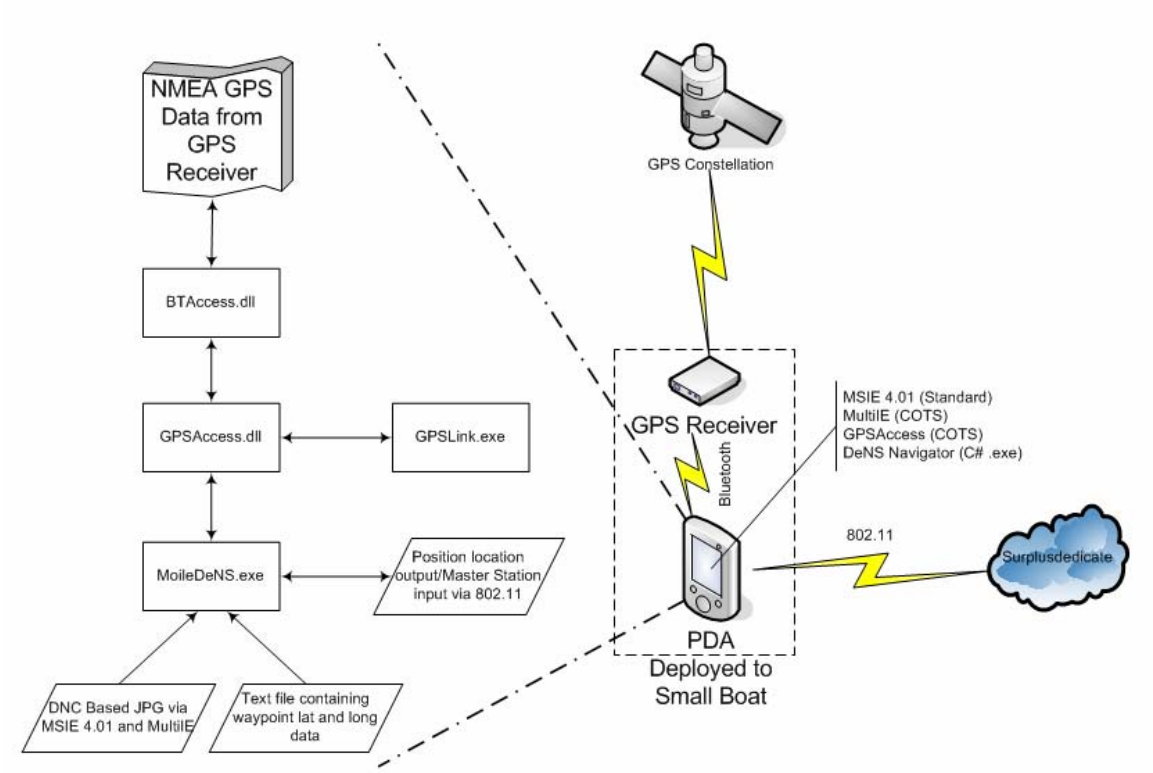

Figure 4. MobileDeNS Architecture

1. Third Party Extensions / Applications Used

There are three third party applications utilized by MobileDeNS: 1) MSIE 4.01; 2) MultiIE; and 3) GPSAccess. The first two will be discussed together, as they function as one program after being installed. The final COTS product will be discussed separately at more length, since it plays a much more central role to the DeNS as a whole.

a. MSIE 4.01 / MultiIE

MSIE 4.01 is the browser that ships with Microsoft Pocket PC 2003 (refer to Table 3 for PDA specifications.) Since Pocket PC and the Compact Framework are both designed specifically to run on a PDA, which is shown to be a limited platform as discussed in Chapter II and demonstrated in Table 4, it follows that many of the features that were not considered critical to functionality were removed in order to improve performance. Likewise, redundant features

(those with multiple paths of execution) were reduced to a single implementation. The result is that many of the features one could expect to readily find on a desktop PC is either not available on a Pocket PC, or if so, they may require some amount of effort on the users part to find. It follows then that the same circumstance would directly apply to MSIE 4.01.

This is the case with a key feature used by MobileDeNS to import the DNC based JPG images that are created through ArcMAP: the ability to save a browsed image to the device. Natively, MSIE 4.01 Pocket PC edition does not support this functionality, thus the MultiIE extension adds it and many others back in with no noticeable degradation in performance. While it is certainly possible to transfer the file via other means such as file synchronization or beaming the file via the IR port, the ability to save the image from a web server was considered less time consuming and more flexible for future work on the prototype system, as will be discussed in more depth in Chapter V.

Beyond the stated ability to save the image to the device, no modification was necessary to either MSIE 4.01 or MultiIE. Since this is the case, further in-depth study of either program is not required. The method of saving the browsed image to the device will be covered in Chapter IV.

b. GPSAccess

GPSAccess is a COTS application that allows the PDA to connect with the Bluetooth GPS receiver and extract NMEA data from it via the provided API. The latest version works only with the Compact Framework V1.0, which requires it be implemented in 2 separate components, GPSLink.exe and GPSAccess.dll. A newer version of the GPSAccess package is currently being developed which will work with the Compact Framework v2.0 which will eliminate the need for GPSLink.exe, but at present, it has not been released (Highpoint Software).

GPSLink.exe is a standalone program which runs on the PDA and works with GPSAccess.dll to provide threading for the GPS connections and data

collections. Additionally, it allows easy monitoring of the connection and its status outside of MobileDeNS if troubleshooting is required in regards to the GPS connection (Highpoint Software). For the DeNS, it should be started prior to launching MobileDeNS. This ensures the proper handling of the GPS connection when the program is launched.

GPSAccess.dll was built on High Point Software's BTAccess library, which is a leading software development kit (SDK) for Bluetooth communications with all Pocket PC devices using the Widcomm/Broadcom Bluetooth stack. GPSAccess utilizes the BTAccess.dll to transparently search for and connect to Bluetooth enabled GPS receivers. It then opens an appropriate COM port, and starts collecting data from the receiver. As such, GPSAccess has the distinction of being the world's only Bluetooth-integrated GPS library for the Widcomm stack, and eliminates many of the otherwise prevalent Widcomm popup screens requiring user interaction. GPSAccess.dll is used by both GPSLink.exe and MobileDeNS via a class called GPS that offers all of the properties and methods that make up GPSAccess, including GPS methods such as GPS.Start() and GPS.Stop(), which starts and stop data collection, respectively, as well as the GPSData class, which is used to extract all GPS data available (Highpoint Software).

2. Architecture Details

MobileDeNS is the PDA application that handles mapping, plotting and transmission of GPS positions as well as other data between the PDA and the Master Station. This section provides detailed architectural insight into how MobileDeNS works and the classes and forms that provide its functionality. Figure 5 provides a block diagram containing all classes and forms used by MobileDeNS. Arrows indicate calling forms and classes, i.e., Form1.cs can call FormMapInfoLat.cs, GPSHandler.cs, and FormMapDetailsTest.cs. Dotted lines are provided strictly for visual guidance, as the structure was manipulated to provide a more efficient fit into this paper.

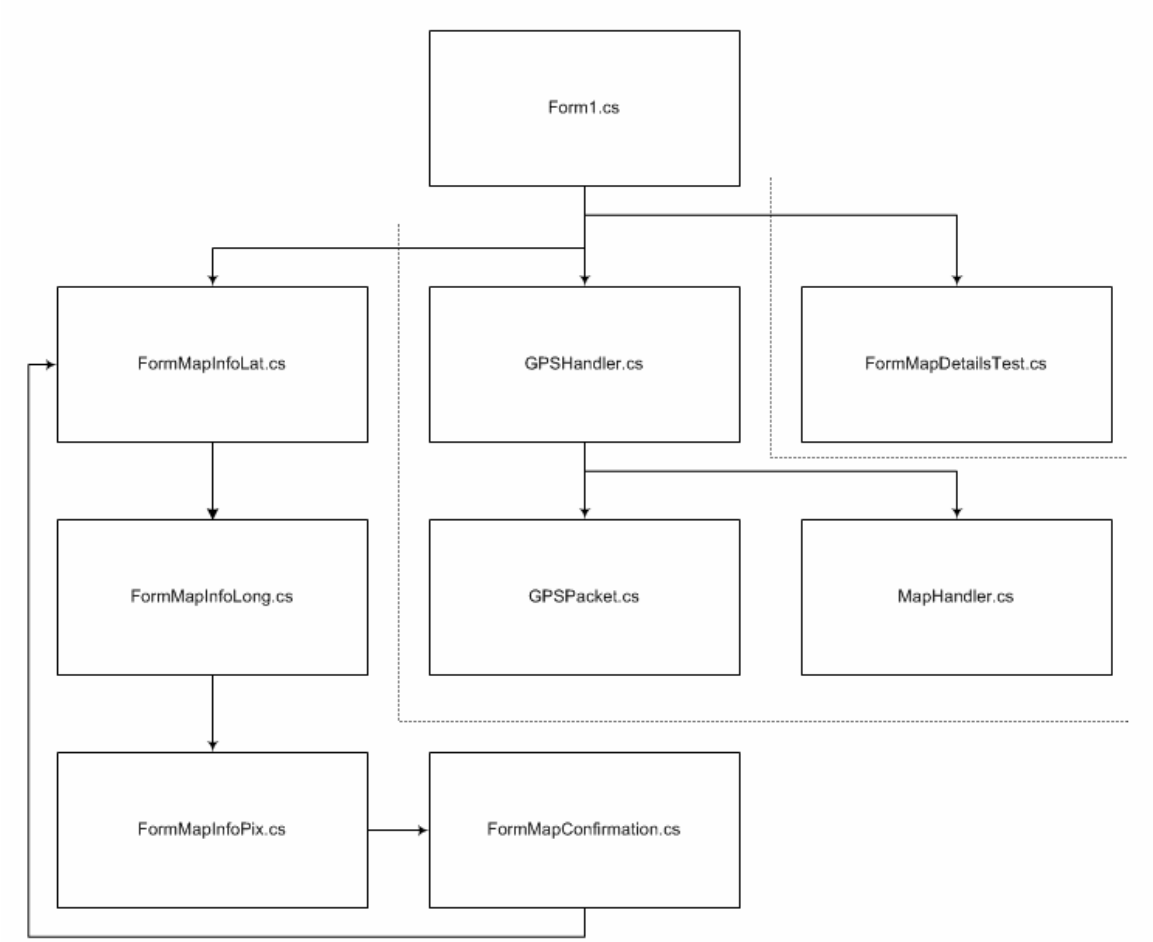

Figure 5. MobileDeNS Class Structure

Pocket PCs use forms to interact with the user, whether that interaction is getting information into the program, displaying information already in the program, or calling other classes to perform certain functionalities within the program. As such, there is typically no main class within a mobile device application. Rather, a distinction is made between form types: a form is either classified as a dialog box, which can be considered a temporary form to get or display information; or a form is considered a main form which serves as the home page throughout the user's navigation of the application (Yao).

It follows, then, that Form1.cs as seen in Figure 5 is a main form. It contains the logic to keep the program running, drawing on all other resources of the application either directly or indirectly to do so. The other five forms shown

are considered dialog boxes, as they are temporary windows that either receive information from, or display information for, the user. Additionally there are three helper classes that handle specific functionalities within the application. Discussion of these components will be accomplished by grouping them into two distinct groups, dialog boxes and classes.

To aid in the logical grouping of all components, Form1.cs will be included in the discussion of the classes, while the remaining forms will be discussed as dialog boxes. The brief descriptions that follow are intended to give a sense of what the various components of MobileDeNS do, not to provide an in-depth understanding of exactly how these functions are carried out. For more clarification on exactly what is taking place within the program, please refer to the appropriate section of Appendix I.

a. Dialog Boxes

(1) FormMapInfoLat.cs. This form allows the user to enter the upper and lower latitude bounds into the system for the map to be displayed. Latitude is entered in degrees, minutes, and seconds (seconds to the nearest hundredth) followed by the one letter designator N or S (North or South, respectively.) The Monterey Bay test map values for all parameters are displayed by default. This form receives the MapHandler instance by reference and writes all values directly to its variables when the OK button is clicked. It is the first in a series of four forms called when loading a new map into MobileDeNS or changing map parameters on a currently loaded map.

(2) FormMapInfoLong.cs. This form allows the user to enter the left and right longitude bounds into the system for the map to be displayed. Longitude is entered in degrees, minutes, and seconds (seconds to the nearest hundredth) followed by the one letter designator E or W (East or West, respectively.) The Monterey Bay test map values for all parameters are displayed by default. This form receives the MapHandler instance by reference and writes all values directly to its variables when the OK button is clicked. It is

the second in a series of four forms called when loading a new map into MobileDeNS or changing map parameters on a currently loaded map.

(3) FormMapInfoPix.cs. This form allows the user to enter the pixel size into the system for the JPG map to be displayed. Pixel size is entered as integers, and can be obtained by viewing the jpg's properties. The Monterey Bay test map values for all parameters are displayed by default. This form receives the MapHandler instance by reference and writes all values directly to its variables when the OK button is clicked. It is the third in a series of four forms called when loading a new map into MobileDeNS or changing map parameters on a currently loaded map.

(4) FormMapConfirmation.cs. This form allows the user to review all the information entered previously (latitude, longitude and pixel size) and then either accept these changes or go back and revise them. This form receives the MapHandler instance by reference and reads all values directly from its variables. This is the last in a series of four forms called when loading a new map into MobileDeNS or changing map parameters on a currently loaded map.

(5) FormMapDetailsTest.cs. This form allows the user to view the latitude and longitude differences currently being used by the application to calculate the pixel position of the position indicator to be drawn on the JPG map. The application uses a series of functions to convert geospatial positions into pixel coordinates for accurate plotting. The latitude and longitude differences being shown are the result of these functions, representing how much arc of both latitude and longitude relates to one pixel on the screen.

b. Classes

(1) Form1.cs. As previously stated, this is the main form that acts as the director for the rest of the application. When this form is running, the application is running as well; likewise, when it has been properly exited (via the exit option on the File menu) the application is terminated. User interaction with the form occurs primarily through the menu system provided along the bottom of the screen, consisting of three main menus: File; GPS; and Map. Each

menu has one or more options that can be selected to drive the program as required by the user. These options will be further discussed in Chapter IV, Section B.

More to the point for the discussion at hand is the functionality taking place that is transparent to the user. When the application is started, immediately after the form is created, it: 1) loads the default map: a DNC based JPG of Monterey Bay, along with all of it's parameters; 2) sets the viewport to orient itself over the upper left-most corner of the underlying JPG (due to screen limitations on the PDA, only 256 x 224 pixels are visible at a time leaving the rest of the JPG off-screen;) 3) an instance of GPSHandler is instantiated; 4) it recognizes that no position yet exists to try to display; 5) it immediately tries to connect to the Master Station server which should be listening on port 5000; and 6) loads the waypoint data from an external file named "waypoints.txt". Action 5 requires that the Master Station be running at the time that MobileDeNS is started for successful completion, as indicated above.

GPS data collection is managed through the use of a timer that polls the GPS receiver for new data every five seconds. If the resulting position is not located on the map as defined by the latitude and longitude bounds entered at the map's load, then that position will not be plotted and the user will be informed that they are not currently in a viewable position. Otherwise, Form1 will use calls to its instance of GPSHandler and subsequent calls from GPSHandler to GPSPacket and MapHandler to calculate the latest position and plot it if it is currently within the viewport. Further, the position will be tested for it's proximity to the designated track, as defined by the waypoints. If it is within tolerance (50 yards for the prototype, though this should be a user assigned distance in the fielded system) it will be plotted using a green positional cursor. Otherwise, it will be plotted using a red cursor to serve as a visual indicator to the PDA user and flags will be sent to the Master Station to indicate that the position has exceeded tolerance.

(2) GPSHandler.cs. This class is responsible for GPS receiver interface and management, as well as interaction with the MapHandler and GPSPacket classes on behalf of Form1. Upon instantiation, this class, in turn, creates an instance of MapHandler and an array to store 2,000 GPSPacket instances. Utilizing these instantiations, GPSHandler can get positional fix details; Parse NMEA data received as strings, and retrieve the pixel X and Y coordinates for a given position. In terms of GPS management, GPSHandler can start or stop the GPS receiver connection, poll the receiver for live data, and determine if the connection still alive. This is also the class that calculates the positions relation to the designated track tolerance mentioned above.

(3) GPSPacket.cs. GPSPacket is instantiated through the use of strings created based on data taken from the NMEA strings received from the GPS receiver. When it is passed the string from GPSHandler, it stores the string as several smaller substrings, and parses numbers appropriately for latter calculations within the application. GPSPacket basically acts as a storage vessel for GPSHandler to use as required to organize positional data for later retrieval. All of its methods are either get- or set-type methods.

(4) MapHandler.cs. While MapHandler does not interact with the DNC based JPG map in any way, it is the class which holds all of the map's parameters for use in calculations. It stores all of the latitudinal and longitudinal boundary information as floats, as well as the latitude and longitude per pixel. JPG pixel height and width is stored in integer form. It is the responsibility of the MapHandler class to calculate the X and Y pixel coordinates of a given latitude and longitude, as well as to determine whether a given coordinate is valid within the bounds of the loaded map.

3. Data Flows

The data flow diagram shown in Figure 6 provides a visual representation of the data flow between those components of MobileDeNS previously discussed. Note that this does not specify control, or how it is passed among the various components. As can be seen, Form1 interacts solely with the Master

Station via the 802.11 network and GPSHandler, which handles the bulk of data flow management. This is achieved through the dialog boxes because as they are spawned by Form1, the instance of GPSHandler created in Form1 is passed by reference to the forms, allowing them to directly update or retrieve the parameters in GPSHandler's instance of MapHandler. FormMapConfirmation.cs, for example, retrieves its information directly from the current values of MapHandler allowing for review of the current parameters at the time it is spawned.

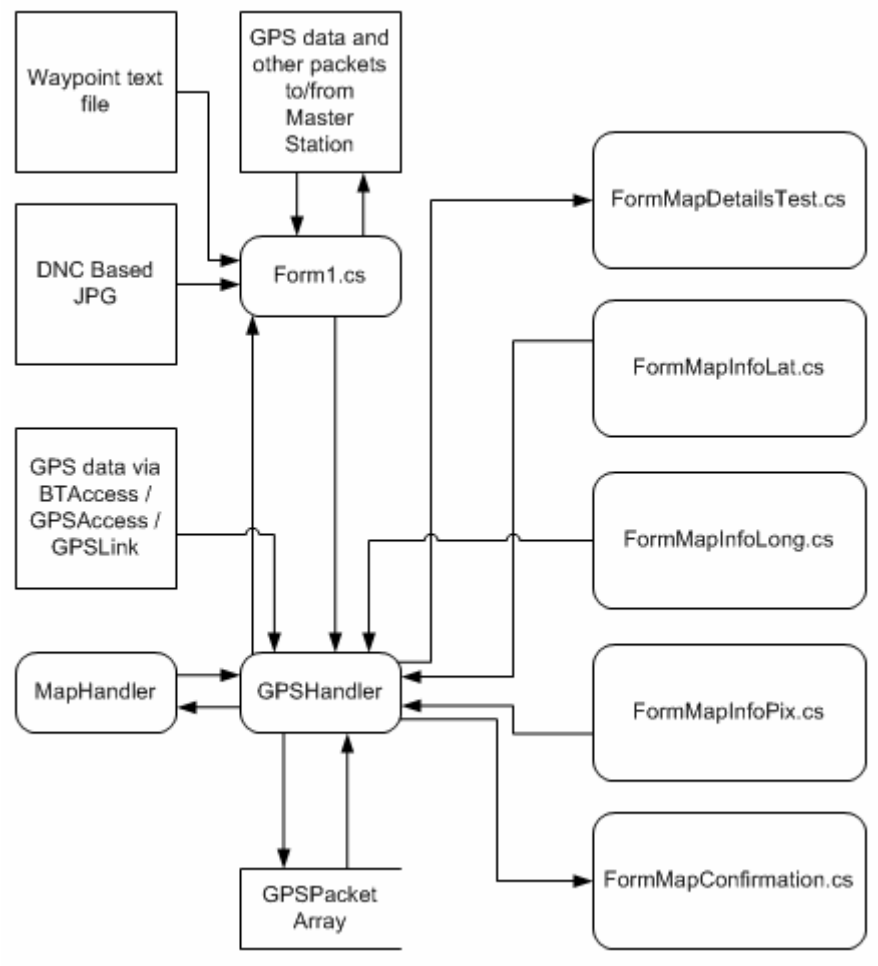

Figure 6. MobileDeNS Data Flow Diagram

As has already been discussed, GPSHandler instantiates MapHandler as well as the GPSPacket instances that are stored in the array, so access to these instantiations is solely available through GPSHandler. Similarly, since

GPSHandler manages the connection to the GPS receiver, it follows that data retrieved from the GPS receiver should be solely available through GPSHandler and its management of the GPSPacket array, as is the case. The DNC based JPG will reside on the PDA device and be made available through the use of Open File dialog boxes; it is considered an external source in Figure 6. Similarly, the waypoint text file must also reside on the PDA, though it is accessed transparently to the user.

D. DeNS

DeNS, as discussed here, is that portion of the system that is intended to remain aboard the parent vessel. It consists primarily of three parts, excluding the Surplusdedicate network which will not discussed further for reasons noted above: 1) the DNC database; 2) the Web/GIS Server; and 3) the Master Station. Figure 7, below, provides an overview of the architecture that comprises this portion of the DeNS prototype system. The blow up of the Master Station, as with the PDA shown in Figure 4, shows the software interaction as well as input and output that is required or produced, respectively. As can be seen, there is no interaction with the GPS which results in a much simpler architecture at the Master Station than was required on the PDA.

The DNC database, having already been discussed in Chapter II will warrant no further direct discussion here. It is, however, important to note the meaning of the lines depicted in Figure 7 connecting the DNC to both the Master Station and the Web/GIS Server. In the case of the Master Station, the solid lines indicate that the DNC data is directly imported and manipulated on the Master Station. The dashed lines between the DNC database and the Web/GIS Server indicate that while the database itself is not necessarily accessible from the server, its use (on the Master Station to create the map file) is a prerequisite to the map product's display by the ArcIMS software.

While the Web/GIS Server plays a small role in the current prototype of the DeNS system, it could potentially take on a much larger, more central role as

will be discussed in Chapter V. Visual Studio .NET 2003 is mainly used on the server to facilitate the editing of web pages at this time. Though this function could just as easily be accomplished through other software that performs the same function, Visual Studio was chosen because of its prior implementation in the construction of MobileDeNS and its potential use in future work as will be discussed in Chapter V. Additionally, IIS, the web server implemented in the solution, can natively interface with Visual Studio, further increasing compatibility throughout the project.

Though ArcIMS can be used with other web server suites, such as Apache Tomcat, IIS was selected due to its integration with Windows and ease of installation. Likewise, the servlet engine, New Atlanta's ServletExec, was selected for its availability; it is provided on the ArcIMS CD. ArcIMS itself consists of three major components: 1) ArcIMS Author; 2) ArcIMS Administrator; and 3) ArcIMS Designer. These components are briefly discussed below in terms of interoperability and function, while more thorough descriptions of their Installation and use can be found by consulting the applicable reference (ESRI, Getting Started with ArcIMS).

ArcIMS Author is the first step in the ArcIMS process to enable maps created in ArcMAP to be delivered via the web. ArcIMS displays files using a configuration file (.axl file) based on an existing shapefile (created in ArcMAP) (ESRI, Getting Started with ArcIMS). Specific guidance on this process can be found in the ArcIMS documentation cited. It is important to note that prior to saving your final .axl file, it is a good idea to review the layer ordering and symbology representation, such as shape, color and pattern to ensure they are still acceptable for proper web display.

Once the .axl file is created, the next step involves ArcAdminstrator. This component of the ArcIMS suite allows ArcIMS services to be created and started in order to serve the .axl files previously created (ESRI, Getting Started with ArcIMS). Once this is accomplished, ArcIMS Designer can be used to build the web site which uses the service to deliver the .axl files (ESRI, Getting Started

with ArcIMS). ArcIMS Designer does this through a series of simple wizard-like screens which require the site's creator to provide minimal information, after which it builds the site automatically. Again further details can be found on these processes in the applicable reference (ESRI, Getting Started with ArcIMS).

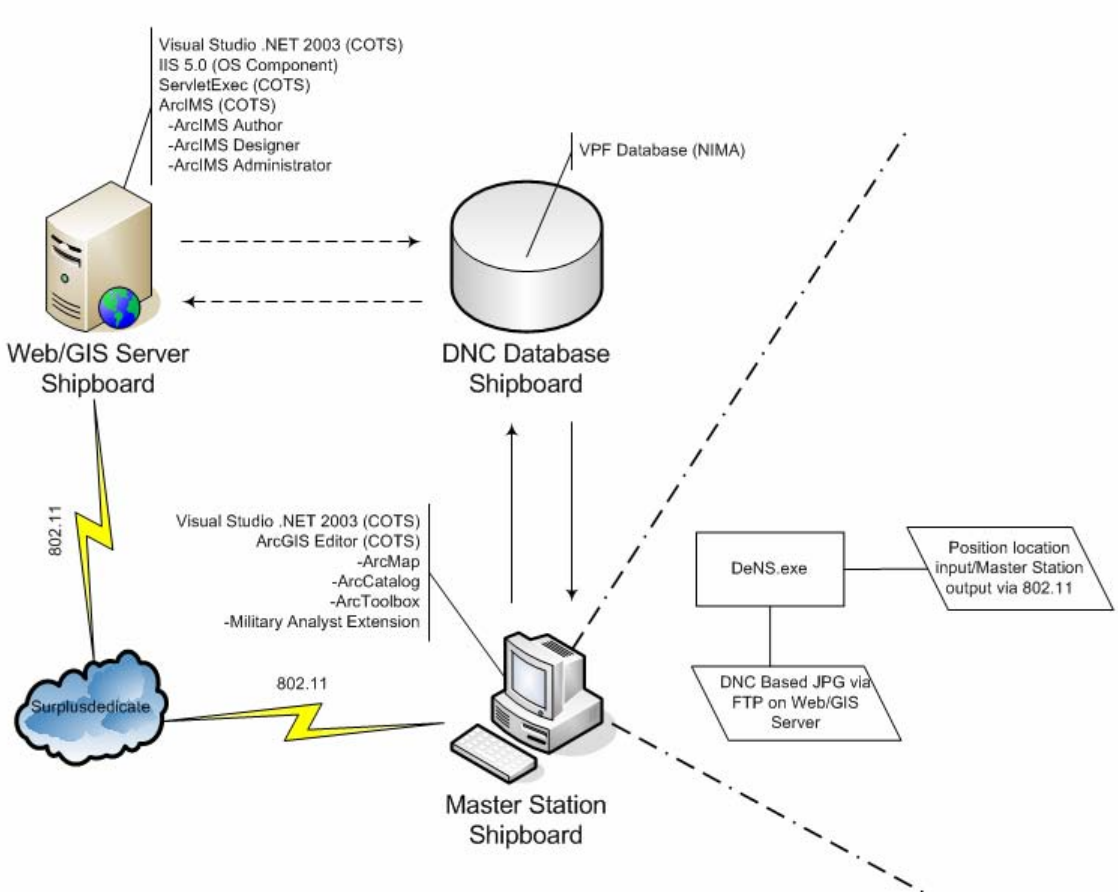

Figure 7. DeNS Architecture

After this entire process is complete, there exists a web site that is accessible via the internet and capable of displaying vector products in the form of .axl files. Unfortunately, the current implementation of the prototype system does not use these .axl files, though that would be desirable and will be further discussed in Chapter V. In order to mitigate this issue, content repurposing was implemented in the web site via Visual Studio. By checking browser versions when a user tries to access the file, the web site can determine if the user may be coming from a PDA (MSIE 4.01) vs. more capable desktop computer (MSIE

5.0 or later.) Based on this observation, it can redirect the PDA user to a site from which they are presented a list of available DNC based JPG maps that can be downloaded to the PDA itself using the MultiIE extension discussed earlier. User's coming from desktop systems running more advanced browsers can choose to view the web presentation of .axl files and/or use FTP in order to download the same JPG files as the PDA user.

1. Architecture Details

DeNS operates on the desktop using the same logic as MobileDeNS with a few exceptions. The positional information that is being plotted comes across the 802.11 network from the PDA, vice a Bluetooth connection to the GPS receiver in the case of MobileDeNS. After the positional data is received, it is handled in the same manner as the GPS data after it is parsed in MobileDeNS. Additionally, DeNS acts as the server for communication, while MobileDeNS acts as the client. This means that when DeNS is started, it immediately starts listening on port 5000 for incoming connection requests from the PDA. Figure 8 provides a block diagram containing all classes and forms used by DeNS. Solid arrows indicate calling forms and classes, while dashed arrows indicate that a class is accessed by reference as will be discussed in Section D.2, below.

As DeNS was ported to the desktop based upon the work completed in MobileDeNS (intended for use on a PDA), much of its functionality is common, though names have changed. There still exists a distinction between main forms, dialog boxes, and classes, and therefore the discussion will follow the same format as that of MobileDeNS above.

Form1.cs is still the main form, containing the logic to keep the program running, drawing on all other resources of the application either directly or indirectly to do so. Due to considerable differences in screen size between the desktop and the PDA, the number of dialog boxes has been reduced from five to two; one for entering map parameters, and one for reviewing map parameters. While the number of forms has decreased, the number of helper classes has

increased primarily due to the addition of the DeNSServer class. This has also resulted in some of the functionality be moved from one class to another, though the logic involved has remained the same.

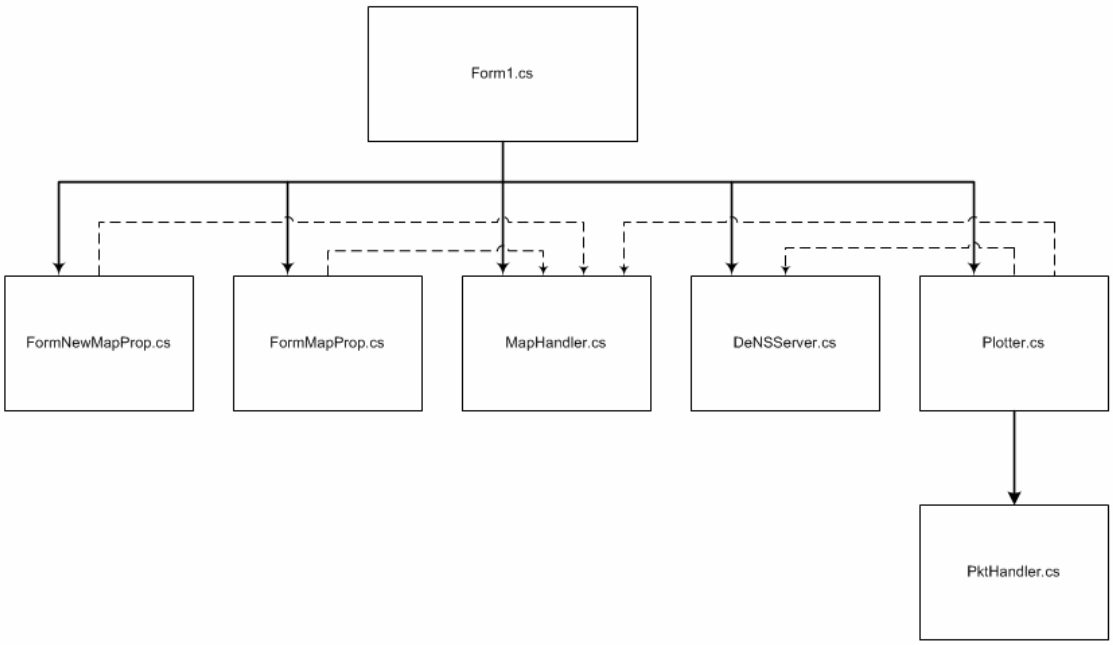

Figure 8. DeNS Architectural Detail

a. Dialog Boxes

(1) FormNewMapProp.cs. In contrast to MobileDeNS, Dens allows the user to enter all required map parameters in one screen. In MobileDeNS, screen size prohibited this ability, which is obviously not an issue with larger 14+ inch screens available for the Master Station on which DeNS runs. Another noticeable difference is the elimination of the requirement to enter the jpg pixel size, which can be gathered automatically by the program through the jpg's properties. Default values for Monterey Bay are provided when the form is instantiated.

(2) FormMapProp.cs. This form allows the user to review the current map parameters, including the latitude and longitude per pixel currently being used by DeNS to calculate the position cursor placement on the DNC based JPG map currently loaded in DeNS. This screen is read only and therefore doesn't allow parameters to be changed.

b. Classes

(1) Form1.cs. As previously stated, this is the main form that acts as the director for the rest of the application. Unlike its MobileDeNS counterpart, Form1.cs when implemented in DeNS can be exited either through the File menu by selecting Exit, or by using the “X” in the upper right-hand corner of the programs main window. The difference is due to the different way that mobile devices handle what is typically considered the close command on a standard desktop computer. The “X” in the upper right hand corner of a PDA application typically only minimizes a window, but leaves the application running, utilized primarily because of screen limitations in an effort to provide the user the opportunity to use more than one program at a time. User interaction with the program occurs primarily through the menu system provided along the top of the screen, consisting of three main menus: File; Map; and Connect. Each menu has one or more options that can be selected to drive the program as required by the user. These options will be further discussed in Chapter IV, Section C.

As with MobileDeNS, there is certain functionality that takes place transparent to the user. When the application is started, immediately after the form is created: 1) it creates an instance of MapHandler and DeNSServer; and 2) The instantiations of MapHandler and DeNSServer are passed by reference to create an instance of Plotter, which can now access the same information and threads that Form1 uses.

As stated previously, DeNS does not interface directly with the GPS receiver, so all data collection takes place via the 802.11 network by way of the DeNSServer instantiation. Positions are plotted when the plotting function is turned on through the Map menu. When activated, a timer is used to request new information from the DeNSServer every five seconds which is then displayed on the DNC based JPG to represent the small boat’s position. Waypoint data is not required by DeNS as all calculations associated with track tolerance take place on MobileDeNS, with indicators being received via the Surplusdedicate network and processed upon receipt.

(2) DeNSServer.cs. This class is the counterpart of GPSHandler found in MobileDeNS. It is responsible for: the management of the 802.11 connection, packet transmission between DeNS and MobileDeNS; the proper handling of transmissions sent and received; storage and retrieval of packets received via the network connection; and proper handling of track tolerance indicator data. Upon instantiation, DeNSServer creates two ArrayLists; one to store position packets, and one to store message packets. It also starts listening on port 5000 for incoming connection requests from MobileDeNS clients.

(3) MapHandler.cs. The MapHandler class in DeNS, as with MobileDeNS, does not interact with the DNC based JPG in any way. It simply acts as a method to load the map, and a vessel in which to store the parameters entered for that map. Unlike its MobileDeNS counterpart, however, MapHandler in DeNS does not perform any calculations nor determine validity of positions, as this functionality has been moved to other methods contained in DeNS.

(4) Plotter.cs. The Plotter class is the class that is responsible for calculating the x and y screen positions for a given latitude and longitude. As such, it is necessary that it can access the DeNSServer for position packet retrieval, and MapHandler for access to the currently loaded map's parameters. To achieve this end, both instantiations are passed by reference to the plotter class at the time of its own instantiation, allowing it to access the most current information when needed. This class is also the class that instantiates PktHandler when a new position is to be plotted for display.

(5) PktHandler.cs. PktHandler is the class which is responsible for all parsing of the position based packets received by DeNSServer. As such, it is also the class which retrieves data from the packets for use in calculations, and user-information oriented display.

2. Data Flows

The data flow diagram shown in Figure 9 provides a visual representation of the data flow between those components of DeNS previously discussed. As with Figure 6, this does not specify control, or how it is passed among the various components. Unlike MobileDeNS, data flow in DeNS indicates a much more dispersed architecture. As previously stated, the logic, (i.e., processes, mathematical equations and data formats) remain consistent between the two applications. This ensures that the same manipulations are taking place on each end system, providing consistency between the two.

What is different is where these manipulations take place, which fundamentally stem from inherent differences between the .NET Framework and the Compact Framework on which Microsoft's C# language is based. As was previously discussed, much of the redundant or superfluous functionality of the full .NET Framework was removed from the Compact Framework in order to reduce it's size from a 40 MB footprint with approximately 86 libraries and 20+ supported languages down to a 2 MB footprint with only 18 libraries and 2 supported languages that would be more compatible with mobile devices (Yao).

After these fundamental differences shaped the architectural changes between the two end applications, other logical structural changes followed. For example, the replacement of the GPSHandler class with the DeNSServer class resulted in the data store for packets existing within the DeNSServer class, forcing the Plotter class to retrieve information as needed and create a PktHandler instance with which with which to manipulate it.

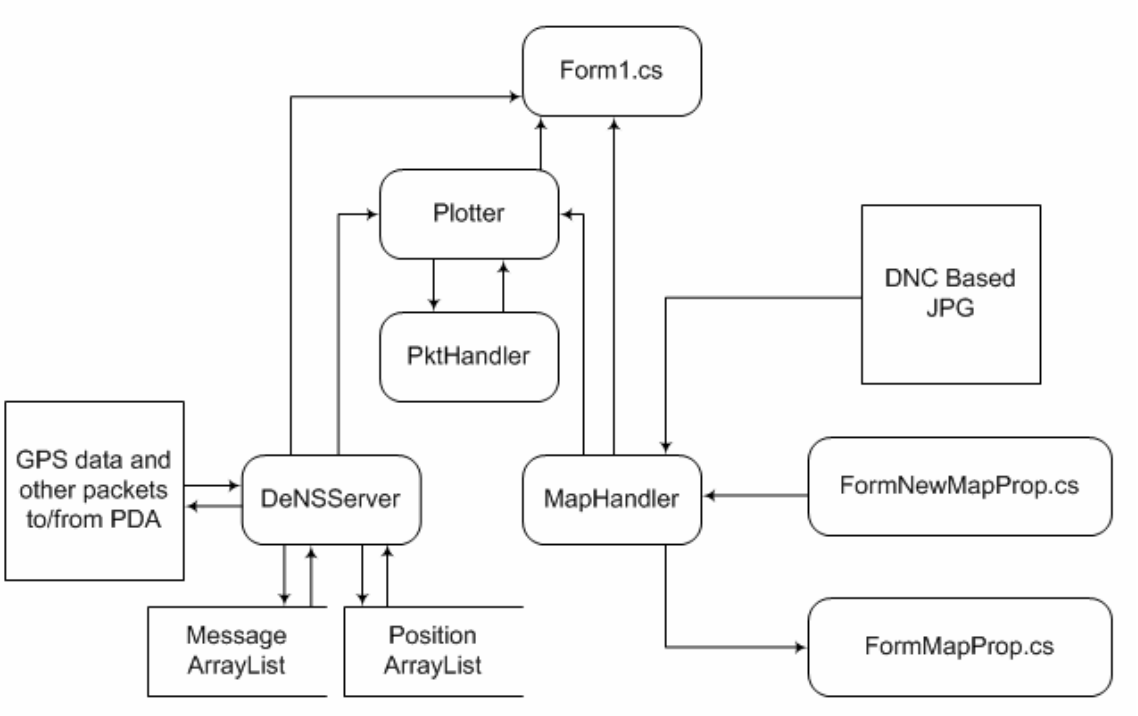

Figure 9. DeNS Data Flow Diagram

E. SUMMARY

Chapter III has provided an in-depth look at the architecture used in the construction of the DeNS prototype system. Beginning with an overall view of the system as a whole and how the various components interact with one another, we then moved to a more detailed view of MobileDeNS and DeNS, respectively. In these more detailed descriptions, we again began with an overall view of the components of the system in question, moving finally to a discussion of programming structure and data flow descriptions. More detailed programming insight can be gleaned from Appendices I and II by reviewing the actual code and the inline comments provided.

In the next chapter, we will discuss the usability issues, functions, and features associated with the DeNS. Also provided will be a discussion of the specific limitations of, and known bugs that exist in the DeNS prototype.

IV. IMPLEMENTATION

A. OVERVIEW

Chapter III discussed the DeNS implementation from an architectural/programming perspective. While Chapter IV also discusses implementation, it approaches the subject from a program usability aspect. In this chapter, both MobileDeNS and DeNS will be examined in terms of usage, features, implemented error handling, and prototype limitations and existing bugs which have been identified but not addressed due to time constraints. The wireless network architecture, setup, and implementation will not be discussed as they are beyond the scope of this thesis.

B. MobileDeNS

As was the case in the previous chapter, MobileDeNS will be discussed first and applies to the application intended for deployment to the small boat, i.e., it will deal with the MobileDeNS application that runs on the PDA. While the GPS receiver is also deployed on the small boat, all of the access and control mechanisms are implemented within the PDA application.

1. Usage

This section will focus on basic functionality and usage of the MobileDeNS program. Discussion will be limited to Installation, proper start-up and shut-down, actions required to begin the navigation process, and basic usage. Specific features implemented in the prototype application will be covered in the next section.

a. Installation

The Installation process of MobileDeNS involves the setup and installation of four distinct pieces of the DeNS system: 1) Wireless setup and

connectivity; 2) MultiIE; 3) GPSAccess; and finally 4) the MobileDeNS application. The installation of all of these components takes place on the PDA.

(1) Wireless setup. As stated above, wireless network connectivity will not be covered except to say that it will be assumed that prior to installation the PDA's networking capability has been established with regards to connectivity to the network medium of choice for DeNS installation (the Surplusdedicate network in the case of the prototype.) Additionally, it is assumed that the Bluetooth capability on the PDA has been activated and is capable of detecting and connecting to Bluetooth enabled devices within the area of the PAN.

(2) MultiIE. With the PDA resting in its docking cradle, run the MultiIE DesktopInstall.exe application. This will Install MultiIE to the desktop computer and subsequently, allow it to be chosen for installation to the PDA via the "Add/Remove Programs…" option available on the Tools menu of Microsoft Activesync if the PDA installation does not start automatically. After installation of MultiIE is complete on the PDA, it must be registered. Without registration, MultiIE is only available for a 15 day trial period, after which its functionality is locked until registration is complete. To register the software, a registration code must be purchased from Southway Software for $15. Upon receipt of the registration code, it and the registering email address must be entered by clicking the register button available from the "About" tab located on the "MultiIE Options…" menu accessible by choosing the MultiIE icon shown in the MSIE window, as per Figure 10.

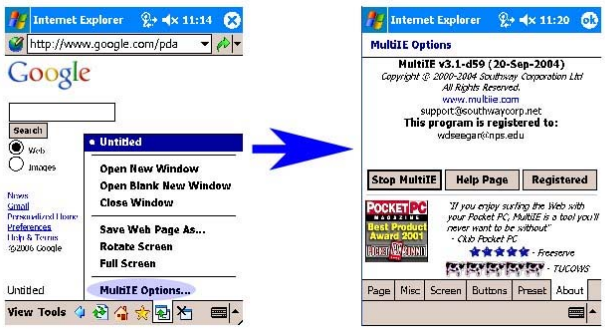

Figure 10. "MultiIE Options…" will provide access to the registration option.

(3) GPSAccess. The zip file acquired from Highpoint Software contains a number of files the must be extracted to the hard drive of the computer prior to installation. After all files have been extracted, the installation process consists of simply copying a selected few of them into the /windows directory on the mobile device. The easiest way to accomplish this is to browse to My Computer\Mobile Device which will be accessible if the device was left properly cradled as indicated above, otherwise the PDA will have to be cradled before progressing with the installation.

There are two BTAccess dll files that were extracted as a result of unzipping the file above, each used to address different versions of the Bluetooth widcomm stack that could be used by the Bluetooth device in question. For the DeNS prototype, we will be using BTAccess4700.dll. To install the GPSAccess application, simply copy BTAccesss4700.dll, GPSAccess.dll, and GPSLink.exe to the same directory as the MobileDeNS application will be installed to (the Program Files directory under its own folder named DeNS is recommended.) Once this is accomplished, rename BTAccess4700.dll to BTAccess.dll. This completes the GPSAccess installation. Further information on the installation process is available in the GPSAccess User's Guide (Highpoint Software), and the readme.txt file extracted from the original zip file.

(4) MobileDeNS. To install MobileDeNS, simply copy the MobileDeNS.exe file, created from the source code located in Appendix I, to the PDA. This can be copied anywhere on the PDA, though for standardization purposes it is recommended that it be placed in the Program Files directory under its own folder named DeNS. The GPSAccess files mentioned above must be located in this same directory.

Before using MobileDeNS, a map will have to be made accessible to it on the PDA. This can be done by downloading a map from the ArcIMS server, detailed later, or by transferring a map using Microsoft Activesynch, beaming, or any other appropriate method. It should be noted that for development reasons, the MobileDeNS prototype attempts to load the MontereyBay.jpg map at startup. If this map is not located in the "My

Documents\My Pictures" folder, startup will fail. For this reason, it is recommended that when installing the prototype system, you copy the DNC based JPG at the time of installation. This is a limitation of the prototype system and should not be carried through to a fielded system, as will be discussed later in this chapter. Additionally, the "waypoints.txt" file must exist in the "My Documents" folder of the PDA prior to launching the program. Again this is a limitation of the prototype system and the fielded system should allow the user to operate without the file, or to browse to the appropriate directory to locate the desired file.

b. Start-up

Once the DeNS installation is complete (consisting of both DeNS and MobileDeNS), MobileDeNS is ready to be started up as follows and can then be expected to function within its normal parameters. MobileDeNS startup is a two part process consisting of activating GPSLink.exe followed by MobileDeNS.exe. As previously discussed, GPSLink is the tool through which MobileDeNS is able to access the GPS data made available from the GPS receiver. As such, it is necessary to have established a connection with the receiver prior to trying to read any data.

Activation of both GPSLink.exe and MobileDeNS is accomplished, as with any other PDA application, by tapping its icon on the PDA screen from the directory location of installation. The use of shortcuts to facilitate easier access to both GPSLink.exe and MobileDeNS may be used. These shortcuts can easily be placed on the start menu for quick access. Additionally, as mentioned above, the MontereyBay.jpg file is required to be in the appropriate directory for the prototype to start-up error-free, as is the waypoints.txt file.

c. GPS Activation

In order to connect to the GPS receiver, simply choose connect from the GPS menu from within MobileDeNS. This will begin the position

collection and display process which will repeat every five seconds in the prototype system, though the fielded system should provide a mechanism to change the timer setting. If the GPS receiver is not currently connected, GPSLink has not been started, or the positions read as the default positions (i.e., 0.0 N by 0.0W) MobileDeNS notifies the user that synchronization has not yet taken place.

While GPSLink is not required to ensure that MobileDeNS will properly start-up, i.e. will not shut down with errors, it must be activated before attempting to connect with the GPS or otherwise a satellite synchronization notification will appear at every GPStimer tick as indicated above. When testing the prototype system, it proved helpful to start GPSLink.exe and monitor the connection status on the PDA, starting MobileDeNS only after the first legitimate position is obtained. The reason for this is that the error control for the GPS connection is based both on GPS connection status *and* the positional information that would indicate that the initial 3-4 minute wait for GPS positional calculation and synchronization are past. While the GPS receiver is connected but awaiting it's first valid positions from the constellation, a default value of 0.000^0 is used for both latitude and longitude. The act of monitoring the status of the GPS connection via GPSLink proved to eliminate the frustration of getting synchronization messages while waiting for the initial position to be obtained. Once the status in GPSLink shows connected and latitude and longitude positions vary from the default position, the GPS receiver has synchronized and is ready for use.

d. Usage

Basic usage of the MobileDeNS application is simple, assuming the prerequisites for running MobileDeNS have been met: maps have been created; a network is established and available; GPSLink is running and providing valid GPS data; and DeNS is running on the Master Station to act as the server side of the system. After the MobileDeNS prototype application has been started, the default map is automatically loaded, and the network connection is established

with DeNS. At this point MobileDeNS is ready to start collecting and displaying GPS positions from the GPS Bluetooth receiver. This is accomplished by selecting “Connect” from the GPS menu within MobileDeNS which starts the GPSTimer that manages the GPS data collection and display process. Once this has been done, the collection and display process, as well as shipping the data back to the Master Station for remote display and monitoring, takes place with no user intervention. Stopping this process is equally simple: choose the “Disconnect” option from the GPS menu within MobileDeNS (formerly the “Connect” option) and the collection, display and relay functions will be stopped. A more in-depth look at some of the features available to the user will be provided in the next section of this chapter.

e. Shut-down

As has been previously discussed, the MobileDeNS application is only properly exited through the use of the “Exit” option on its File menu. Due to the differences in mobile devices and desktop computers, using the “X” in the upper right hand corner simply minimizes the form and leaves the application running, using valuable resources on the PDA, perhaps unintentionally. Using this minimize feature for switching to other applications, such as GPSLink to verify or monitor the GPS receiver connection, is particularly handy when troubleshooting. It is also possible that future fielded versions of DeNS will have the capability to send and receive other file types, such as image files, documents, video etc. If this is the case, the minimize feature will prove its worth in allowing access to other programs to import/export, edit, and/or otherwise manipulate these file types.

2. Features

In the last section general usability of the MobileDeNS application was provided. This section elaborates on the usability of the application by providing insight into the various features and functionality that it contains. All menus will be discussed first, along with the various features accessible through them.

Following the menu discussions, access methods and discussions of each feature available through other means will be given. Any associations to limitations and known bugs will be provided for reference in both instances, though specifics of said limitations or bugs will be discussed in the appropriate section of this chapter.

a. File Menu

The file menu in the prototype system consists of only one available option, Exit. As this option has already been discussed above, no further discussion is necessary in this section. It should be noted that while the File menu in the prototype system only contains one option, the File menu in the fielded system will probably contain more. As with the other menus to be discussed, those options envisioned for the fielded system will be visible but disabled. This ensures that the vision of the system at inception will be considered and either carried out or discarded in the final product.

b. GPS Menu

Like the File menu, the GPS menu in the prototype system has only one available option, Connect. This option allows the MobileDeNS application to connect to the GPS receiver via the Bluetooth connection and GPSAccess.dll. Additionally, it changes this particular menu item's text to "Disconnect", and uses the state of the displayed option's text to determine what logic to implement; connection logic, or disconnect logic.

Based on the option's text at selection, the menu item will call the start or stop function of the GPSAccess.dll, change the text display to either connect or disconnect as will be appropriate for the next selection of this menu item, and either enable or disable the GPSTimer to manage the collection, display, and relay of GPS data in the appropriate fashion.

c. Map Menu

The Map menu in the prototype system is where most of the available menu options reside. It consists of four available options: 1) Clear Map; 2) ***Test***; 3) New Map; and 4) Properties. Each of these available options will be discussed in more depth below.

(1) Clear Map. This option allows the current position to be erased from the map's display, but does not alter the internal mechanisms used to store either the said position, or the currently loaded map parameters. It is intended as a method to clear the map of positional displays, which is admittedly not of substantial use in the prototype, but will be of much greater use in a fielded system which should be capable of displaying track histories.

This will become apparent if, for example, a small boat is deployed along a predetermined track and ordered to linger in one area for a prolonged period. If old positions are being maintained within the DeNS, then multiple positional displays will begin to overwrite one another, potentially causing a great amount of confusion in sorting out which is which. In this case, the ability to clear the map of positional data, while maintaining the program's integrity in regards to track history becomes paramount. This feature allows the map to be cleared as described above, but continue tracking the small boat's progress unimpeded. One note: as the prototype was designed to only display one position at a given time, logic will have to be built in to allow multiple positions to be cleared. This can easily be accomplished through the use of an appropriate data structure and the logic to loop through that data structure setting all or some of the positions properties to not visible.

(2) ***Test***. This option was used primarily for many different troubleshooting tasks during the prototype design, hence the lack of an otherwise descriptive handle. Its final use, however, was for the display of latitude and longitude per pixel; a feature that proved invaluable in verifying proper plotting of GPS positional data received. As such the decision was made to leave this feature in the prototype system as a reference for further development in the fielded system. It is not anticipated that this information

would prove useful to the average user of a fielded system, so it should be considered for removal prior to fielding the system, though if sufficient justification is present, it may included. Fields displayed are read only, so no changes to internal parameters are possible through this display.

(3) New Map. This feature allows the user to load a new map and change the map parameters to conform to said map. Upon selection, this option will open a file dialog box that will allow the user to browse to a specific directory from which to load a new map file. As has been previously stated, certain limitations exist on PDA devices that do not exist on desktop computers. One such limitation is the depth to which a user can browse through an open dialog box. On a desktop computer, the user can browse to an unlimited depth, however, through a mobile device (running the Pocket PC OS), that depth is limited. To alleviate this issue, the decision was made to place map files in the My Pictures directory of My Documents, a second level directory that is both reachable and readily familiar to typical PC or Pocket PC users.

Based on the above decision, the initial directory for the open file dialog box is the “\My Pictures” directory on the PDA, since the PDA is intended in the DeNS system for exclusive use within the system. This, of course, can be changed in the fielded system if desired but it is recommended that an equally high level directory be used to skirt the issues associated with lower level directories in terms of browsing. In any case, once the file has been chosen through use of the dialog box, it will be loaded into the viewport for display, and the sequence of specifying the map’s parameters via the forms detailed in Chapter 3.C.2.a will begin. It should be noted here, that if the user chooses the “OK” button on FormMapConfirmation.cs, all parameters will remain loaded and the map will be ready for use. If, however, the user spots an error made in entering the data, choosing the “Change” button will allow them to go back to the beginning of the parameter specification process and enter the correct information.

(4) Properties. The Properties option allows the user to review all map parameters previously entered for the current map, and either

accept them or change them as outlined directly above. Choosing this option places the user on the FormMapConfirmation.cs form; the end of the map parameter form cycle. From this point they can choose to either accept the displayed parameters, or revise them as described above. This option is intended to allow review and correction to the currently loaded map parameters at any time.

d. View Fix Details

Certain details of any given fix can be viewed when desired by tapping on the fixes positional cursor displayed on the current map. Each fix cursor is actually an active button. The action of clicking on a given fix calls the button's action listener and will result in a message box with the fix's time, latitude and longitude. Currently there is a formatting bug in this portion of the code that results in a somewhat garbled, confusing display as will be discussed at more length in Section 4 below.

e. Auto-scroll

At the time of design, it was decided that an "auto-scroll" feature that automatically scrolled the viewport in order to center the current position would be a desirable feature to implement. This would allow the user to watch the orientation of the viewport as their current position was plotted by allowing smooth transitions to the current position. This feature is automatic and cannot be turned off. For reasons discussed in section B.4 of this chapter, however, this same feature is listed as a bug in the prototype system.

f. Fix Distance from Track

This feature compares each GPS fix position to a designated track as defined by the waypoints.txt file. In the prototype system only two waypoints were used to create and test this feature, though the fielded system should allow for multiple waypoints, thus enabling multiple legs of a single track to be

traversed with tolerance indicators provided. Figure 11 provides a graphical depiction of the triangle used to determine whether the fix position is within the allotted track tolerance. Distance in the DeNS system is determined using trigonometry to derive the height of the tolerance triangle, represented in Figure 11 by the line segment CD.

In order to determine the length of segment CD, the length of the triangle's side, a, b, and c must be known. This information is derived using features provided by GPSAccess, specifically the GetVector method that is available by enabling Geo-fencing within the application. This method uses two latitude and longitude coordinate pairs, or simply put two positions. By using both waypoint coordinates for the track and the GPS fix in question (annotated as A, B, and C respectively in Figure 11), three GPSvectors are returned from corresponding calls to the GetVector method. Each of these GPSvectors represents one side of the triangle, depending on the parameter combinations that were passed, and each instance of this class contains a property for the vectors length.

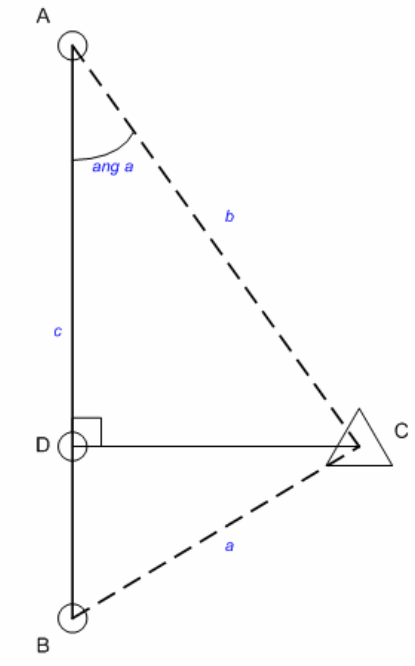

Figure 11. Track Tolerance Triangle

Once the lengths are made accessible, they are used mathematically to determine the cosine of angle a (represented by “ang a” in Figure 11) using the Law of Cosines provided in Figure 12. Since all three sides of the triangle are known at this point, the equation is manipulated algebraically, as shown, so that it can be used to derive the cosine of “ang a”. This manipulation allows all known measurements to be applied in order to produce the cosine of “ang a” to which the arc cosine can then be applied to produce the actual measurement of “ang a”.

$$a^2 = b^2 + c^2 - 2bc * \cos(\text{ang a})$$

$$\frac{a^2 - b^2 - c^2}{-2bc} = \cos(\text{ang a})$$

Figure 12. Law of Cosines Restated

After the measurement of “ang a” has been determined, the triangle shown in figure 11 can be reduced to the smaller triangle ACD for the remainder of the calculation. This results in the hypotenuse of the new triangle being the line segment CA, while the opposite side from “ang a” becomes line segment CD. Since the measurement of the angle has already been derived, the trigonometry function stated in Figure 13 can be manipulated as shown to solve for the distance of the fix for the track.

$$\sin(\text{ang a}) = |CD| / |CA|$$

$$|CD| = |CA| * \sin(\text{ang a})$$

Figure 13. Sine Function Restated

Once the length of line segment CD is known it is compared to the track tolerance being used by the program to determine if the fix has exceeded the tolerance. If it has not, then the position will be plotted in green, otherwise, the position will be plotted in red. Either way, a packet is sent back to the master station along with the position to signify to DeNS if the position is within tolerance so that it too can process the position accordingly. Finally, if the new fix has exceeded tolerance a message packet will also be sent to the Master Station for display to the user informing them of the deviation from track.

3. Error Handling

It is important to note that the DeNS, as has already been stated numerous times, is a prototype system. As such, much more attention has been placed on proper operation and handling of valid data, than on ensuring data entered is valid. It was assumed in the creation of the prototype that all users would enter information that was correct, with very little validation taking place. The majority of prototype data validation and error handling occurred in the areas that were beyond the limits of programming control, i.e. the TCP/IP connection via the implemented wireless network and the GPS connection and data. Of course, when fielding the system this will not be the case. In a fielded system, extensive validation and error handling would have to be implemented to ensure the safety of the boat and its crew. Below are the validation and error handling methods that were implemented in the Prototype system.

a. Socket Loss to Master Station

The user of the DeNS system is expected to have specific knowledge in setting up and establishing the network medium of choice for implementation. Proper setup of the network will negate half of the potential error messages a user will encounter, as these messages will either be of the nature of network resources not being available, or even in the event of proper setup and implementation, an interruption in service resulting in loss of the established connection.

The former, network resources not available, is typically a result of improper setup in routing and/or port forwarding or port triggering if an internal router is being used. This can also result if the DeNS application is improperly shut down and still has an instance running in the background unbeknownst to the user. In this case accessing the Task Manager in the Master Station and the Running Programs window on the PDA will allow the rogue instances of the program to be stopped, and the resources consumed to be released for reuse. As previously stated, it is assumed that the user has an intimate knowledge of setting up network services, and as this endeavor is beyond the scope of this thesis, due to the innumerable network configurations available and the prototype nature of the project at this time, the former will not be discussed further.

While the later instance, a previously successful connection that has subsequently failed, has not happened in testing of the prototype system, it is possible given power fluctuations, variations in visibility of the network, etc. If this occurs, there is currently no way to successfully recover in the prototype system, though a message box will be displayed informing the user that an error reading the server data has occurred. In this case, the prototype system should be shut down along with the master station and restarted. The fielded system should implement a graceful way in which to reestablish the connection without requiring shutdown.

b. GPS Connection Interruption

If, during the operation of the MobileDeNS application, the GPS connection fails it can attributed to one or both of two reasons, either the Bluetooth connection is no longer valid, or the position reads 0.000^0 latitude by 0.000^0 longitude (the default reading). The former can occur if the connection has been lost, which occurred in testing, due to failure of the GPS device due to insufficient battery to sustain the connection, or if entering a GPS-denied coverage area, i.e. connection to the GPS constellation was not accessible due to occlusion, or both.

As has been previously discussed, either occurrence will result in "Waiting for Satellite Synchronization" messages. While the issue of constellation occlusion can occur at any time for a myriad of reasons, it is not expected to occur at great periodicity at sea where the system is expected to be fielded, as access to satellite coverage should be unimpeded as long as the PDA and GPS units remain under open sky. On the contrary, the default reading of 0.000^0 latitude by 0.000^0 longitude occur whenever the GPS receiver is trying to synch up with the GPS constellation and obtain an initial position. This is the fundamental reason for starting GPSlink.exe, and monitoring the progress of said link, prior to starting the MobileDeNS prototype. In the fielded system, there should be a mechanism in place to monitor the connection from the application, ensuring adequate synchronization before the boat is deployed.

c. Position Location Outside of Map Parameters

As has been established, the prototype system was designed to operate within a specific map (Monterey Bay) and its associated parameters at any given time. As such, logic was implemented which would result in any GPS position that was either above the upper latitude or the left longitude limits, and/or below the lower latitude or right longitude limits, be flagged as an invalid position; the user would be notified that they were no longer within the limits of the map currently loaded, and the position would not be plotted. That said, the limits of the map currently being displayed are the responsibility of the user, and accuracy and attention to detail in entering said limits is paramount to the accurate operation of the DeNS. In the fielded system, these same logic operations should be applied universally to current map parameters, regardless of the map's coverage area.

d. Unreliable GPS data

According to NMEA standards, reliable GPS data is annotated by the 38^{th} character in the NMEA string. It should be noted that as with any string in the C# programming language, the first character is considered to be the 0^{th}

character, thus the fix quality character in the NMEA string would be accessed by specifying the 37th character. This character can range from 0 – 8 inclusive, with associated meanings as defined in Table 6 below.

Designator	**Meaning**
0	Fix not available
1	GPS fix
2	DGPS fix
3	PPS fix
4	Real Time Kinematic
5	Float RTK
6	Estimated (dead reckoning)
7	Manual input mode
8	Simulation mode

Table 6. NMEA Fix Quality Indicator Definitions.

From these definitions, it is obvious that we are only interested in those NMEA strings with a designator of one; a GPS fix. Therefore any NMEA string with a designator other than 1 is discarded by the GPSHandler class. The user is then notified that a bad packet was received, and the plotting, display, and relay of said packet does not occur.

4. Limitations and Known Bugs

As with most prototype systems, the DeNS components were built within certain pre-conceived parameters. These parameters imposed limitations on the functionality and effective areas of operation of the prototype system. Additionally, certain known bugs exist within the program that were not able to be

corrected due to time limitations. This section lists both limitations and bugs that are known to exist within the prototype system along with the justification of why the limitations were imposed or the suspected cause of the bug as applicable.

a. Prototype Operation Limited to 36^{o} N and 121^{o} W

The DeNS prototype was developed as a proof of concept prototype for C2 during over the horizon small boat operations at NPS in Monterey California. In order to speed development by simplifying the coding logic, the decision was made to limit the effective area to the degrees of latitude and longitude above. The result of this design decision is that only minutes and seconds are used to calculate fix positions on the screen. This does not present a problem for the prototype system, as all map files implemented are constrained to the same coordinate areas, with no overlap to the next higher or lower degrees allowed. As has been discussed previously this is included in the assumption of accurate user-provided data for the system to operate with. Naturally a fielded system would have to expand to be operable at all possible latitudes and longitudes with potential overlaps in order to be considered a viable solution, as the US Navy conducts operations around the world.

b. Prototype Operation Limited to NW Hemisphere

It follows from the discussion of latitude and longitude constraints above that the DeNS will only operate in the NW hemisphere of the globe. It is important to note, however, that aside from the necessary calculations for overlapping degrees of latitude and longitude within the same hemisphere, considerations must be made when applying the same calculations in different hemispheres, and still other considerations exist when the area of coverage requires that hemisphere boundaries be crossed.

In the case of areas of coverage within the same hemisphere (i.e. both upper and lower boundaries of coverage are either North or South, and both left and right boundaries are either East or West) It is a simple case to determine

how to properly perform the mathematic application of the spatial differences and the upper and left boundaries in order to properly calculate the pixel position of the fix for display to the screen. Noting that the upper left corner of the map is always pixel position (0, 0), while the same corner represents an arbitrary latitude and longitude which will either be greater than or less than the lower right corner depending on the hemisphere, one can readily see how it will be necessary to apply the spatial difference in order to calculate the pixel position of a given GPS fix. For example, In the Northern hemisphere the spatial difference must be subtracted from the upper bound because a valid GPS fix for the given map parameters will always be less than the upper bound. That difference is then converted into the appropriate number of pixels to be plotted on the screen. The reverse is true for the Southern hemisphere, however, in which case the valid GPS fix will always be greater than the map parameters upper bound, necessitating that the upper bound be subtracted from the fix position in order to generate the difference for pixel conversion and subsequent plotting. The same is true for the Eastern and Western hemisphere.

The case of overlapping hemispheres creates a new problem; the GPS fix position's hemisphere must first be determined with relation to the upper and left boundaries of the map parameters. If the position lies in the same hemisphere as the boundary, the calculation can take place as described above. If, however, the position lies across the boundary of the hemisphere then two calculations must take place, one to determine the spatial difference between the hemisphere boundary and the map parameter boundary in question, and another to determine the spatial difference between the GPS fix position and the hemisphere boundary. The resulting two differences are then added together to obtain the total spatial difference of the GPS fix position from the map parameter boundary in question, either upper or left. That difference is then converted to pixel coordinates for screen display.

c. DeNS Application on the Master Station Must be Running

As previously mentioned, DeNS performs all server functions from the Master Station. In the prototype system, the TCP/IP socket connection was programmed for automatic connection at startup. While this sped up development of the prototype system, it requires that DeNS be running and awaiting incoming connection requests before MobileDeNS is launched in order to prevent Mobile DeNS from shutting down due to an unsuccessful connection. In the fielded system it is recommended that this feature be replaced by the option to start the connection on both sides of the system when the user desires, which will eliminate this problem and expand the functionality of the PDA application to areas beyond the scope of small boat C2, as will be further discussed in Chapter IV.

d. GPSLink on PDA Must be Running

As previously stated, this is a requirement imposed by the GPSAccess.dll. While a new version of GPSAccess is in progress that will alleviate this requirement and allow connection to the GPS receiver directly from within MobileDeNS, it has not yet been released.

e. Default Map Required

As was discussed earlier in both this chapter as well as Chapter 1, the MobileDeNS prototype was designed with the constraint of operability within the Monterey Bay area exclusively. As such, in order to speed development and testing, the decision was made to automatically load the Monterey Bay DNC based JPG map file on application start-up. This necessitates the map file be located in the appropriate directory as discussed in the installation section of this chapter. In the fielded system, this requirement should be removed by loading a map file at the users request rather than automatically doing so at start-up.

f. Waypoint File Required

This limitation has also already been discussed. The waypoint file is required by the prototype system at application start up in order to read in track information for processing. It is a simple text file that provides latitude and longitude coordinates in the form DDMMSS.SSDDDMMSS.SS where D is degrees, M is Minutes, and S is seconds using decimal notation with two significant decimal digits, and Latitude precedes longitude. As mentioned earlier, only two waypoints can be used in the prototype, but the fielded system should allow for waypoint exclusion or multiple waypoints to be used.

g. Fix Details Display

The feature described in section B.2.d above, allows the user to view details of the fix by tapping the fixes positional cursor on screen which is actually a small button that is moved as new positions are read in and displayed. As was discussed, there is currently a bug in the display code which results in a garbled display. While the labels of time, latitude, and longitude are displayed, the data following the labels is not always correctly presented, presumably due to the way that the data is accessed based on the status of the GPS receiver. This is considered to be a minor bug, as the prototype system only displays one position at a time, always the latest one received. As a result, the decision was made to focus on other, more important features of the prototype. In the fielded system, this bug should be eliminated.

h. Auto-scroll

The auto-scroll feature was initially added to the system as a user-enhancement to improve the user perception of bearings and placement on the currently loaded map file when the fix position moved off screen. In the initial version of MobileDeNS, when new fixes were plotted that were off screen, the application would simply jump to a view in which that position was centered. This resulted in a fair amount of user confusion and the decision to make the program

gracefully scroll to the new fix position to allow the user to follow along and avoid any momentary disorientation during use. This feature is automatic and cannot be turned off.

In implementing this feature, however, cascading issues have changed the picture somewhat. What was initially implemented as a user-enhancement, created a situation that in most cases frustrated the user by implementing a slow moving perspective, which at close interval fix times, could take multiple fixes to correctly orient the map. Further issues arose after the auto-scroll feature was added with regards to allowing the user to scroll the map at will, as it would immediately begin to try to reorient itself after the current position was no longer displayable on the screen. Additionally, while the program is performing auto-scroll, it is inaccessible to user interaction.

Based on these results it is recommended that the fielded system either have a method for activating and deactivating the auto-scroll feature, or that it is removed from the system entirely. Alternative methods to avoid the disorientation that precipitated the addition of this feature should be explored, such as the possibility of using lines to connect current positions with their predecessors to indicate the direction of movement on the map along with the area of the map previously visited. Similarly, the GPS information can be used to determine direction and speed, and DR symbology could be used to indicate where the next fix is expected to appear.

C. DeNS

Having discussed MobileDeNS in terms of usage, features, error handling, and limitation and known bugs, we will now do the same with DeNS: that portion of the prototype system the resides on the small boat's parent vessel. As discussed in Chapter III, this portion of the system consists primarily of four separate components: 1) the DNC database; 2) the Surplusdedicate network; 3) the ArcIMS Server; and 4) the Master Station. For the same reason given in Chapter III, the DNC database and the Surplusdedicate network will not be

discussed at any further length, leaving the remainder of this chapter to focus on the ArcIMS server and the Master Station.

1. ArcIMS

The first of these components, the ArcIMS server, will be discussed in a cursory manner, as most of the technical information is available in the applicable reference (ESRI, ArcIMS9: Getting Started with ArcIMS). This document covers basic installation, features, usage, and other technical aspects of the program at great length. What will be discussed here are the different roles, and the methodologies of implementation of said roles, of the ArcIMS server.

a. Map File Server

The primary use of the ArcIMS sever is to facilitate the downloading of the previously prepared DNC based JPG map files to both the MobileDeNS PDA application and DeNS running on the Master Station. For the PDA, downloading is accomplished via the content repurposing strategy previously discussed. Since the PDA browser is not capable of handling the Dynamic Hypertext Markup Language (DHTML) code used by the ArcIMS server, a home page was implemented that checks the browser version and directs requests from MSIE 4.01 to an alternate site from which the appropriate map file can be downloaded. Figure 11 shows the redirection that takes place based on browser version where the black lines indicate the actual redirection, while the blue line indicates the reason for said redirection.

The Master Station can download the same DNC based JPG files from the same directory as the PDA by browsing to it and saving the file from the browser, or alternatively using an FTP site to accomplish this same purpose. Similarly, a link to the applicable directory could be easily added to the home page to ease the facilitation of this process. The link from the home page, shown in Figure 11, will direct the Master Station to a web-based viewer that displays the web-version of the map file created in ArcMap as described below.

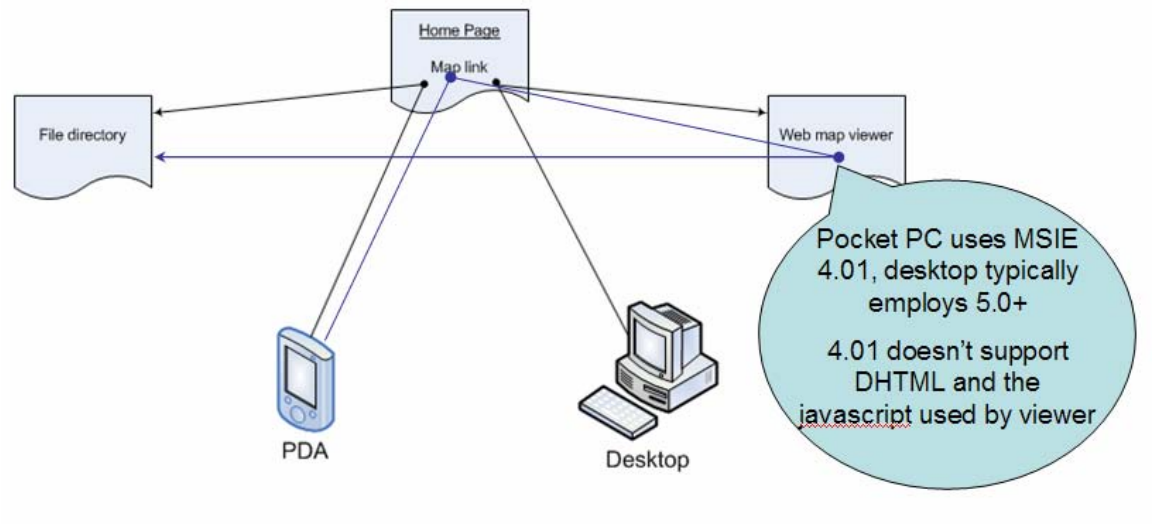

Figure 14. Desktop vs. PDA Access to Map Files

b. Web-based DNC Access

From the Master Station, following the link on the home page places the browser at a web-based viewer that uses the .axl files created in ArcAuthor which retain the underlying DNC database structure. This allows resizing, zooming, panning, and similar functions to be accessed through the browser, as well as selection of specific objects represented on the chart in order to view their underlying characteristics. For example, an object that represents a buoy can be selected and thereby enable the type, color, name, etc, of the buoy to be viewed. Further, from this web-based viewer, zooming in on the map results in its natural de-cluttering as objects are linked to their spatial grid reference not a pixel location. This can be invaluable in helping the small boat, which doesn't have this capability, gain general orientation and also confirmation of those objects which they have depicted on their jpg representation of the chart. Further discussion and application of this aspect of ArcIMS viewer will be discussed in the Future Works section of Chapter V.

2. Master Station

As with MobileDeNS, full explanations for usage, features, error handling, and limitation and known bugs will be provided for DeNS which runs on the

Master Station. As was demonstrated in Chapter III, DeNS is a much simpler implementation, architecturally speaking, than is MobileDeNS. As such, the discussion that follows does not have as many components as did MobileDeNS, though since DeNS was created based on the functionality of MobileDeNS, many of the same issues are likely to exist and will be discussed here as they relate to the Master Station.

a. Usage

This section will focus on basic functionality and usage of the DeNS program on the Master Station. Discussion will be limited to Installation, proper start-up and shut-down, actions required to begin the navigation process, and basic usage. Specific features implemented in the prototype application will be covered in the next section.

(1) Installation. The Installation process of DeNS on the Master Station involves the setup and installation of only two distinct pieces of the DeNS system, as opposed to the four required for MobileDeNS: a) Wireless setup and connectivity; and b) the DeNS application.

a) Wireless setup. As with MobileDeNS, wireless network connectivity will not be covered except to say that it will be assumed that prior to installation, the Master Station's networking capability has been established with regards to connectivity to the network medium of choice for the DeNS installation, the Surplusdedicate network in the case of the prototype.

b) DeNS. To install DeNS, copy the DeNS.exe file (created from the source code located in Appendix II) to the Master Station. This can be copied anywhere on the Master Station, though for standardization purposes it is recommended that it be placed in the Program Files directory under its own folder named DeNS. Unlike MobileDeNS, DeNS doesn't attempt to load a map at startup. This negates the requirement to have a map available before its first use, as was the case with MobileDeNS on the PDA.

(2) Start-up. While MobileDeNS required both itself and DeNS to be installed with DeNS running prior to start-up, DeNS does not have such a requirement. As DeNS implements the server side of the system, it can be started independent of MobileDeNS, and will immediately begin listening for incoming connection requests on port 5000. While DeNS is listening for incoming requests, it is capable of loading maps along with the associated map parameters. This is recommended before starting MobileDeNS in order to ensure that the application is ready to begin receiving and processing GPS data from MobileDeNS, and in any case is necessary before the GPS connection within MobileDeNS is activated.

Activation of DeNS is accomplished, as with any other desktop application, by double clicking its icon on the screen from the directory location of installation. The use of shortcuts to facilitate easier access may be used as the user desires. Once DeNS is running, it is ready to establish the TCP/IP socket connection and begin processing GPS data received from said connection. Additionally, as mentioned above, a map file is necessary to facilitate proper handling of GPS data received, and should be identical to the map and parameters being used in MobileDeNS.

(3) Usage. Basic usage of the DeNS application is simple, assuming the prerequisites for running it have been met: i.e. maps have been created and are available and coordinated between DeNS and MobileDeNS; a network is established and available; and MobileDeNS has been installed and is ready for use. As described above, after the DeNS prototype application has been started, it will immediately begin listening for connection requests from the MobileDeNS client application and allow the user to load appropriate map data. After the map data has been loaded, and the connection is established, the application will begin processing incoming GPS data after the user selects the Start Plotter option from the Map menu. This will continue to plot positions received in accordance with the plotTimer which is set to activate every five seconds. If no new position exists at that time, the user will be alerted, and offered the option to either continue with or stop the plotter’s operation.

It is recommended that the plotter be started in conjunction with the GPS Connect function of MobileDens to ensure that fix position plotting on both ends are being synchronized between the two applications. While time constraints prohibited its implementation in the prototype system, it is recommended that the fielded system automate the start plotter function whenever a new position is received from MobileDeNS, or an alert of some type is implemented that alerts the Master Station that new positional data has been received so the plotter can be started manually.

(4) Shut-down. Unlike its MobileDeNS counterpart, DeNS can be properly exited through the use of the “Exit” option on its File menu, as well as by using the “X” in the upper right hand corner of the application window. As previously discussed, mobile devices and desktop computers utilize the “X” in different manners.

b. Features

In the last section general usability of the DeNS application was reviewed. This section elaborates on the usability of the application by providing insight into the various features and functionality that it contains. Unlike the MobileDeNS discussion above, this section will focus solely on menu driven features. This is because in DeNS there are no features that are available outside of the provided menus. The auto-scroll feature was not added due to issues already cited, and the Fix details feature was similarly excluded due to the faulty logic that resulted in the garbled appearance of intended data. As before, any associations to limitations and known bugs will be provided for reference in both instances, though specifics of said limitations will be discussed in the appropriate section of this chapter.

(1) File Menu. The available file menu in DeNS is identical to that of MobileDeNS, consisting of only one option, Exit. This option operates in an identical fashion to the corresponding option in MobileDeNS, and as this option has already been discussed above, no further discussion is necessary in this section. It should be noted, however, that while the File menu

in the prototype system only contains one option, the File menu in the fielded system will probably contain more. As with the other menus to be discussed, those options envisioned for the fielded system will be visible but disabled. This ensures that the vision of the system at inception will be considered and either carried out or discarded in the final product.

(2) Map Menu. The Map menu in the prototype system is where most of the available menu options reside. It consists of four available options: a) Clear Map; b) Start Plotter; c) New Map; and d) Properties. Each of these available options will be discussed in more depth below.

a) Clear Map. This option allows the current position to be erased from the maps display, but does not alter the internal mechanisms used to store either the said position, or the currently loaded map parameters. It is intended as a method to clear the map of positional displays, which is admittedly not of substantial use in the prototype, but will be of much greater use in a fielded system which should be capable of displaying track histories, as discussed in MobileDeNS above.

b) Start Plotter. This option is used to manually start and stop the plotter function of DeNS on the Master Station. When selected it changes the displayed text of this option to either “Stop Plotter” or “Start Plotter” and either starts or stops the plotTimer, as is applicable. PlotTimer is the counterpart of GPSTimer in the MobileDeNS application and manages the plotting function of all received GPS positions when enabled. When disabled, GPS positions are still received and stored, but are not plotted until the plotter is enabled.

c) New Map. This feature allows the user to load a new map, and change the map parameters to conform to said map. Upon selection, this option will open a file dialog box that will allow the user to browse to a specific directory from which to load a new map file. The browsing limitations which affect the open file dialog box on the PDA do not impede the same dialog box on the Master Station. Similarly, on the desktop it is possible to

automatically retrieve the map file height and width so these parameters are not required from the user when loading the map. Remaining parameters are entered through a single dialog box which is editable so review can take place at the same time as data entry.

d) Properties. The Properties option allows the user to view all map parameters currently entered for the map being displayed. This is a read-only form and does not provide a mechanism to change incorrect parameters. To do this, the user must reload the map using the New Map option from the Map menu.

(3) Connect Menu. The Connect menu provides two available options: a) Test Server; and b) Start Server. In the prototype system, both of these options were used strictly for troubleshooting and testing functionality as described below.

a) Test Server. This option was used early in the development process to ensure that plotting functions were operating correctly. It does not interact with the TCP/IP socket connection in any way, but instead simulates it by storing three NMEA strings in the ArrayList to be used by the plotter. This option is now obsolete and marked for deletion.

b) Start Server. This option does not initiate the server class to begin listening for a TCP/IP connection as the name would imply. Rather, it was used in testing the GPS data retrieval and plotting function of DeNS as a precursor to implementing the plotTimer. It was used after the Test Server option above. This option is also marked for deletion.

(4) Distance from Track. This feature is simplified as compared with the case of MobileDeNS. This is because all of the calculations take place within MobileDeNS, with indicator packets being sent to DeNS to signify whether the latest fix has exceeded the allowable track tolerance. Once the comparison has been made within MobileDeNS, a packet containing either a “W” (within tolerance) or an “X” (exceeded tolerance) is sent to the Master Station. Upon receipt the Master Station then sets a flag to indicate what the fix

cursor color should be for the next fix plotted. Additionally, if the fix has exceeded track, a message indicating such is sent to the Master Station for display to inform the monitoring user of the deviation from track. The prototype implementation of this feature resulted in the possibility of misrepresentation of data as will be discussed in section d below.

c. Error Handling

As was stated when discussing error handling in MobileDeNS, the DeNS is a prototype system and operates under a number of assumptions. As these assumptions have already been listed in the section address error handling for MobileDeNS they will not restated here. The removal of the GPS interface in the Master Station (which receives its data via the 802.11 network) in turn simplified the error handling that was implemented, resulting in error handling primarily concerned with the TCP/IP socket connection. Again, when fielding the system this will not be the case. In a fielded system, extensive validation and error handling will have to be implemented to ensure the safety of the boat and its crew. Below are the validation and error handling methods that were implemented in the Prototype system.

(1) Socket Loss to MobileDeNS. As has been stated previously, the user of the DeNS system is expected to have specific knowledge in setting up and establishing the network medium of choice for implementation. Proper setup of the network will negate half of the potential error messages a user will encounter, as these messages will either be of the nature of the network resources not being available, or even in the event of proper setup and implementation, an interruption in service resulting in loss of the established connection. Both of these cases have been covered under the MobileDeNS section of this chapter, and apply equally to the Master Station.

(2) Position Location Outside of Map Parameters. This error should not occur on the Master Station, since positions outside of the MobileDeNS map parameters are not processed, nor, therefore, relayed to the Master Station. It is implemented in order to bring to light erroneous map

parameters existing between the two end systems. Proper setup of the map and its associated parameters, as well proper coordination between the two applications in regards to the correct data entry for parameters will ensure that this error is mitigated on the Master Station. If it occurs, the map parameters on both applications should be compared and corrected.

d. Limitations and Known Bugs

As with MobileDens, the scope and parameters of the overall project imposed limitations on the functionality and effective areas of operation of the prototype system. Additionally, certain known bugs exist within the program that were not able to be corrected due to time limitations. This section lists both limitations and bugs that are known to exist within the prototype system along with the justification of why the limitations were imposed or the suspected cause of the bug as applicable.

(1) Prototype Operation Limited to 36^{o} N and 121^{o} W. This limitation of the DeNS prototype system applies universally to all components thereof. As such the causes and discussion provided in Section B.4.a of this chapter have not changed and apply here as well. Please refer said section for review of this limitation.

(2) Prototype Operation Limited to NW Hemisphere. As with item 1) above, this limitation of the DeNS prototype system applies universally to all components thereof. As such the causes and discussion provided in Section B.4.b of this chapter have not changed and apply here as well. Please refer said section for review of this limitation.

(3) Coordination Between Plotter and GPS Connect. As mentioned briefly in the section c.4 above, it is imperative that the plotter in DeNS and GPS connect in MobileDeNS be started together. Otherwise lag time could result in misrepresentation of the fixes relation to track when received. For example, if more than an entire fix interval is skipped between the two functions start times, the boat could deviate from track and correct, but the Master Station would see the fix that exceeded tolerance as valid while it was alerted that the fix within tolerance had exceeded it. This compounds as more fix intervals are

skipped. In the fielded system it is recommended that either the synchronization be automated by automatically starting the plotter when fix data arrives, or by adding the tolerance information to the fix packet that is sent to DeNS to ensure it is processed along with the fix to which it applies.

D. SUMMARY

Chapter IV took the architectural information provided in Chapter III, and added to it in terms of program implementation and installation, usability, features, error handling, and limitations and known bugs for each component of the DeNS. While initially focused on those components implemented on the PDA and deployed to the small boat, discussion then moved to components implemented on the parent vessel. The first component reviewed on the parent vessel was the Web/GIS Server providing a very brief discussion with applicable references noted for further in-depth study. Discussion then focused on the Master Station implementation following the same format as that used for discussion of the PDA implementation.

Chapter V will conclude discussion of the DeNS prototype system. It will provide information concerning project conclusions, performance, and test results, and provide suggestions for future works involving the system with the ultimate goal of a fielded system.

THIS PAGE INTENTIONALLY LEFT BLANK

V. DISCUSSION

A. OVERVIEW

Chapters I through IV have provided thorough discussion of the problem addressed by the DeNS prototype system, related background information, and detailed aspects of its implementation in terms of both architecture and usability. In Chapter V, conclusions will be drawn based on the observed results of the DeNS prototype and recommendations for future work and expansion of the system will be given. Finally, the DeNS project will be summarized in terms of functionality, applicability, and performance.

B. CONCLUSIONS

As has been previously stated, the DeNS prototype system was designed to function in the vicinity of the Naval Postgraduate School campus, i.e., in latitude and longitude ranges contained within 36^0 N and 121^0 W. This worked well in the development and testing of the system because the network infrastructure was already in place to facilitate the TCP/IP-based communication between the various components, and the close proximity to the Pacific Ocean. Additionally, the DNC, as with paper charts, provides some coverage of terrestrial areas as well navigable waters. This resulted in the entire campus and surrounding area being available for testing purposes. It should be noted that all testing of the prototype system has taken place on land, rather than in the maritime environment in which it is intended to operate. This should pose no issue, however, in its operation in such an environment, since all charting was derived directly from the same source as would have been used in a maritime environment covering the same area (Monterey Bay.)

The geographic coordinate to pixel coordinate conversion that takes place is considered sound due to the fact that the chart jpg files preserve the scaling factor used by the DNC. Therefore, any given jpg will cover an area represented by a given range of latitude and longitude coordinates without distortion, thus

allowing a pixel to represent a precise amount of latitudinal and longitudinal area. That said, the amount of latitudinal and longitudinal area covered by a pixel was hard-coded into the DeNS applications for use with the test map, just as the degrees for both coordinates and the hemisphere were disregarded, since it was given that it would be contained on the chart and the chart would fall within the same degree ranges. This allowed more rapid development of the prototype system than would otherwise have been possible, though it severely limits usage outside of the designated area defined above.

An additional limiting factor which affected testing was somewhat spotty coverage of the Surplusdedicate network available in an outdoor situation which was necessary for the GPS receiver to maintain satellite connectivity. In testing MobileDeNS as a standalone product, this was mitigated, and proper functionality of the plotting procedure was verified using various landmarks provided on the DNC based jpg image. Likewise, by developing a client tool which allowed positional and message data to be entered and passed to the Master Station in the same format as used by MobileDeNS, it was possible to verify the same GPS and landmark positions, thus proving proper plotting functionality in DeNS, as well as the proper handling of messages.

Once independent verification of both applications was completed satisfactorily, a test path was determined on which all connectivity was possible (Surplusdedicate in an outdoor environment, Bluetooth connectivity between the GPS receiver and the PDA, and GPS satellite connectivity.) Using this path as a test track, both applications were tested and positional data was verified as having been correctly plotted and displayed correctly in regards to track tolerance. As has been previously stated, when testing the applications together in this manner, care had to be taken to coordinate the GPS Connection function on MobileDeNS with the Start Plotter function on DeNS in order to ensure that the track tolerance information displayed was relevant to the current position being displayed. Proper synchronization in this manner allowed verification of the proper functioning of the DeNS prototype system as a whole.

Performance of the system was as expected, i.e., real time information was displayed on MobileDeNS and immediately passed back to DeNS where identical information was displayed for remote monitoring, also in real time. No unexpected issues such as system failure, unknown bugs, or program crashes occurred during at any time during testing of the final prototype system. As such, performance of the DeNS prototype can be characterized as outstanding within the specifications in which it was designed to function.

C. FUTURE WORK

The DeNS prototype system offers an in-road into C2 of small boat operations that has yet to surface in the US Navy. Through future work and expansion of the DeNS project, a myriad of possibilities exist that would not only allow OTH operations at a level previously unattained in terms of safety and precision, but could also extend the use of the MobileDeNS application to other endeavors outside the realm of small boat operations. This section will delineate several ideas for future expansion and use of the system, though it is not meant to be an exhaustive list.

1. Full DNC Implementation

As it currently exists, the DeNS prototype system utilizes jpg based images that are derived from the DNC. While this provides the required minimum level of navigational information to allow the system to operate, it does not exceed that standard. Additionally, it can be a somewhat daunting task to create the charts to be used by system to an average user. In lieu of using the jpg images, work should be conducted to implement the DNC directly on the PDA.

By implementing the DNC on the PDA, it would enable tracks to be updated “on the fly” by the Master Station to allow for dynamically changing missions, or conversely by the small boat based on situational awareness that the Master Station may not be privy to. Additionally, PDA implementation of the

DNC would allow the small boat to query the DNC directly to determine what chart symbology means when required. This could in the form of pulling navaid characteristics for positional verification, or viewing safety warning for specifically charted areas, without requiring messages or voice communications with the Master Station and/or control vessel.

A more flexible display would also result from full DNC implementation, since charts would not be designed with specific boundaries. Instead, charts would be both scrollable and scalable. In the first case, small boat operations would not be constrained by the predetermined jpg image, in which the edge represents a boundary beyond which operations cannot proceed with the DeNS system. When the edge is reached on a DNC implementation, the map would simply scroll to reveal the new area. In the second case, the DNC would allow a scalable image which can be freely zoomed in or out on to provide the necessary view for the situation. Additionally, unlike jpg or bitmap files, when the DNC is zoomed in or out on, it naturally de-clutters because the symbology is not tied to pixel location, but rather a reference location. This allows the area covered to change, while the symbology remains at a constant scale.

In light of all this, it is evident that DNC usage on the PDA is a desirable feature. One potential method of implementation of this feature would be the expanded integration of the ArcGIS line of software into the DeNS prototype system. ESRI, the makers of ArcGIS, also make a PDA application called ArcPAD. ArcPAD is a mobile GIS solution that integrates database access, mapping, and GPS functionality with a multilayer support environment. It supports file formats which include jpg, bitmap, ESRI shapefiles created by ArcMAP, and ArcIMS image Services as was discussed in Chapter IV. Use of this solution could enable both the Master Station and PDA to share a picture being maintained by the ArcIMS server that is already being used for image file access. The entire range of ArcGIS products is customizable and programmable through a C# API extension usable in Microsoft Visual Studio.

2. Network Architecture

As has already been discussed numerous times, 802.11 is not an acceptable medium for use in the fielded system, and should be replaced by a more robust architecture. 802.16 has been presented as the alternative of choice in this paper, though testing in the maritime environment at applicable ranges would have to be conducted before it is considered as the replacement standard. Other thesis projects are currently being conducted on campus into various network architectures on which the DeNS system could be an excellent candidate. One such project is COASTS, being conducted by the Operations Research department. It is an 802.16 based network that has achieved ranges out to 30NM.

3. Program Expansion

The DeNS prototype system lends itself to expansion in many areas. Some of these have already been discussed, such as the ability to start and stop the client/server communication channel and the ability to automatically synchronize the GPS connect and Plotter Start functions. Others are hinted at by currently disabled menu options, such as the ability to send a message packet, load a saved track file, turn on or off tracking features that would display all fixes obtained thus far, find a given fix or set of coordinates, and mark a given position on the display.

Other expansion could also be desirable, such as the ability to send a file along with the coordinates at which it was sent. This could enable, for example, a picture of jettisoned contraband and its position to be saved for later retrieval of said contraband. Similarly, other packet types could be implemented to provide proper handling of data transmitted, such as pictures, scanned documents, and even new map files. Additionally, the infrastructure is already in place to allow text messaging between the MobileDens and DeNS applications, only the user interface is required to enable this feature.

Further expansion is possible when considering applications of the system in operations outside of the small boat arena. Consider the navigation detail in which it is standard practice to issue paper “chartlets,” typically a photocopy of the chart on which navigation is taking place for visual aid to the cognizant people in the navigation detail. Rather than providing a cumbersome stack of paper that must be flipped through by the Conning Officer, MobileDeNS could be issued in a standalone version that would display the GPS position and update the display accordingly, providing real-time information regarding the ship’s position in a channel.

D. SUMMARY

The DeNS prototype system is a viable solution that can be used in a myriad of ways and can be applied to any navigational evolution that takes place aboard a naval vessel. The C2 capability and real-time positional display it provides, coupled with its small footprint and use of use mark it as a much needed asset to the ship’s company. Further development resulting in fleet wide implementation would provide a much needed tool to any platform on which it is used.

APPENDIX I

A. OVERVIEW

Appendix I is intended to provide code listing for forms and classes used in the implementation of the MobileDeNS client of the DeNS prototype system. Each form or class will be detailed separately, with comments provided in-line. All code was programmed in the Microsoft C# language constrained by the Microsoft .NET Compact Framework, and designed to run on a windows based PDA platform utilizing the .NET Compact Framework.

1. Form1.cs

```
using System;
using System.IO;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
using System.Globalization;
using GPSAccess;
using System.Net.Sockets;
using System.Threading;

namespace MobileDeNS
{
    /// <summary>
    /// Main form for MobileDeNS PDA application.  It instantiates
    /// the GPSHandler, GPSpacket, and MapHandler, classes and
    /// contains all menus and the GPSTimer that manages the
    /// polling of and data extraction from the GPS receiver.  It
    /// also contains the threading logic that handles communication
    /// back to the Master Station running DeNS.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Form refFormMapInfoLat;
        private System.Windows.Forms.Form refFormMapConfirmation;
        private System.Windows.Forms.Form refFormMapDetailsTest;
        private System.Windows.Forms.MenuItem FileMenu;
        private System.Windows.Forms.MenuItem GPSMenu;
        private System.Windows.Forms.MenuItem MapMenu;
        private System.Windows.Forms.MenuItem FileMenuExit;
        private System.Windows.Forms.MenuItem GPSMenuDemo;
        private System.Windows.Forms.MenuItem GPSMenuConnection;
        private System.Windows.Forms.MenuItem GPSMenuTracking;
        private System.Windows.Forms.MenuItem GPSMenuDetails;
        private System.Windows.Forms.MenuItem MapMenuOverview;
        private System.Windows.Forms.MenuItem MapMenuMark;
```

```
private System.Windows.Forms.MenuItem MapMenuMarkCurPosit;
private System.Windows.Forms.MenuItem MapMenuMarkCoord;
private System.Windows.Forms.MenuItem MapMenuFind;
private System.Windows.Forms.MenuItem MapMenuFindCoord;
private System.Windows.Forms.MenuItem MapMenuFindLastPosit;
private System.Windows.Forms.MenuItem MapMenuFindCurrPosit;
private System.Windows.Forms.HScrollBar hBar;
private System.Windows.Forms.VScrollBar vBar;
private System.Windows.Forms.Panel FillerPnl;
private System.Windows.Forms.PictureBox viewPort;
private System.Windows.Forms.MainMenu mainMenu1;
private System.Windows.Forms.MenuItem GPSMenuSeperator;
private System.Windows.Forms.Button curPosition;
private System.Windows.Forms.Timer demoTimer;
private System.Windows.Forms.MenuItem MapMenuClear;
private System.Windows.Forms.MenuItem MapMenuTest;
private System.Windows.Forms.MenuItem MapMenuNew;
private System.Windows.Forms.MenuItem MapMenuProperties;
private Thread readThread;
private TcpClient client;
private NetworkStream stream;
private BinaryWriter writer;
private BinaryReader reader;
private int msg = 1;
private string srvMsg = "";
private GPSHandler GPSman;
private System.Drawing.Point upperLeft;
private StreamReader sr = new StreamReader(
    @"My Documents\waypoints.txt");
private String gpsLine;
private bool positExists;
private System.Windows.Forms.Timer GPSTimer;
private System.Windows.Forms.MenuItem FileMenuLoad;
private System.Windows.Forms.MenuItem FileMenuConnect;
private System.Windows.Forms.MenuItem FileMenuSend;
private System.Windows.Forms.MenuItem FileMenuSendFile;
private System.Windows.Forms.MenuItem FileMenuSendMsg;
private System.Windows.Forms.MenuItem FileMenuSendQMsg;
private bool userchange = true;

/// <summary>
/// Form1 constructor
/// </summary>
public Form1()
{
    //Build the form
    InitializeComponent();

    //load the NPS map image
    viewPort.Image = new System.Drawing.Bitmap(
                        "\\My Documents\\My Pictures\\" +
                        "MontereyBayWithTrack.jpg");

    //set the viewport corner coordinates
    upperLeft = new Point(0,0);

    //initialize scrollbar values
```

```
        vBar.Value = 0;
        hBar.Value = 0;

        //instantiate GPSHandler class
        GPSman = new GPSHandler();

        //No posit exists yet
        positExists = false;

        //connect to DeNS on Master Station
        readThread = new Thread(new ThreadStart(runClient));
        readThread.Start();

        //Load waypoint data from file
        getWaypoints();
    }

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    protected override void Dispose( bool disposing )
    {
        base.Dispose( disposing );
    }
    #region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        this.mainMenu1 = new System.Windows.Forms.MainMenu();
        this.FileMenu = new System.Windows.Forms.MenuItem();
        this.FileMenuExit = new System.Windows.Forms.MenuItem();
        this.GPSMenu = new System.Windows.Forms.MenuItem();
        this.GPSMenuConnection =
            new System.Windows.Forms.MenuItem();
        this.GPSMenuTracking =
            new System.Windows.Forms.MenuItem();
        this.GPSMenuDetails =
            new System.Windows.Forms.MenuItem();
        this.GPSMenuSeperator =
            new System.Windows.Forms.MenuItem();
        this.GPSMenuDemo = new System.Windows.Forms.MenuItem();
        this.MapMenu = new System.Windows.Forms.MenuItem();
        this.MapMenuOverview =
            new System.Windows.Forms.MenuItem();
        this.MapMenuClear = new System.Windows.Forms.MenuItem();
        this.MapMenuMark = new System.Windows.Forms.MenuItem();
        this.MapMenuMarkCurPosit =
            new System.Windows.Forms.MenuItem();
        this.MapMenuMarkCoord =
            new System.Windows.Forms.MenuItem();
        this.MapMenuFind = new System.Windows.Forms.MenuItem();
        this.MapMenuFindCoord =
            new System.Windows.Forms.MenuItem();
        this.MapMenuFindLastPosit =
```

```
        new System.Windows.Forms.MenuItem();
    this.MapMenuFindCurrPosit =
        new System.Windows.Forms.MenuItem();
    this.MapMenuTest = new System.Windows.Forms.MenuItem();
    this.MapMenuNew = new System.Windows.Forms.MenuItem();
    this.MapMenuProperties =
        new System.Windows.Forms.MenuItem();
    this.hBar = new System.Windows.Forms.HScrollBar();
    this.vBar = new System.Windows.Forms.VScrollBar();
    this.FillerPnl = new System.Windows.Forms.Panel();
    this.viewPort = new System.Windows.Forms.PictureBox();
    this.curPosition = new System.Windows.Forms.Button();
    this.demoTimer = new System.Windows.Forms.Timer();
    this.GPSTimer = new System.Windows.Forms.Timer();
    this.FileMenuLoad = new System.Windows.Forms.MenuItem();
    this.FileMenuConnect =
        new System.Windows.Forms.MenuItem();
    this.FileMenuSend = new System.Windows.Forms.MenuItem();
    this.FileMenuSendFile =
        new System.Windows.Forms.MenuItem();
    this.FileMenuSendMsg =
        new System.Windows.Forms.MenuItem();
    this.FileMenuSendQMsg =
        new System.Windows.Forms.MenuItem();
    //
    // mainMenu1
    //
    this.mainMenu1.MenuItems.Add(this.FileMenu);
    this.mainMenu1.MenuItems.Add(this.GPSMenu);
    this.mainMenu1.MenuItems.Add(this.MapMenu);
    //
    // FileMenu
    //
    this.FileMenu.MenuItems.Add(this.FileMenuExit);
    this.FileMenu.MenuItems.Add(this.FileMenuLoad);
    this.FileMenu.MenuItems.Add(this.FileMenuConnect);
    this.FileMenu.MenuItems.Add(this.FileMenuSend);
    this.FileMenu.Text = "File";
    //
    // FileMenuExit
    //
    this.FileMenuExit.Text = "Exit";
    this.FileMenuExit.Click +=
        new System.EventHandler(this.FileMenuExit_Click);
    //
    // GPSMenu
    //
    this.GPSMenu.MenuItems.Add(this.GPSMenuConnection);
    this.GPSMenu.MenuItems.Add(this.GPSMenuTracking);
    this.GPSMenu.MenuItems.Add(this.GPSMenuDetails);
    this.GPSMenu.MenuItems.Add(this.GPSMenuSeperator);
    this.GPSMenu.MenuItems.Add(this.GPSMenuDemo);
    this.GPSMenu.Text = "GPS";
    //
    // GPSMenuConnection
    //
    this.GPSMenuConnection.Text = "Connect";
```

```
this.GPSMenuConnection.Click +=
    new System.EventHandler(this.GPSMenuConnection_Click);
//
// GPSMenuTracking
//
this.GPSMenuTracking.Enabled = false;
this.GPSMenuTracking.Text = "Tracking";
//
// GPSMenuDetails
//
this.GPSMenuDetails.Enabled = false;
this.GPSMenuDetails.Text = "Details";
//
// GPSMenuSeperator
//
this.GPSMenuSeperator.Text = "-";
//
// GPSMenuDemo
//
this.GPSMenuDemo.Enabled = false;
this.GPSMenuDemo.Text = "Demo";
this.GPSMenuDemo.Click +=
    new System.EventHandler(this.GPSMenuDemo_Click);
//
// MapMenu
//
this.MapMenu.MenuItems.Add(this.MapMenuOverview);
this.MapMenu.MenuItems.Add(this.MapMenuClear);
this.MapMenu.MenuItems.Add(this.MapMenuMark);
this.MapMenu.MenuItems.Add(this.MapMenuFind);
this.MapMenu.MenuItems.Add(this.MapMenuTest);
this.MapMenu.MenuItems.Add(this.MapMenuNew);
this.MapMenu.MenuItems.Add(this.MapMenuProperties);
this.MapMenu.Text = "Map";
//
// MapMenuOverview
//
this.MapMenuOverview.Enabled = false;
this.MapMenuOverview.Text = "Overview";
//
// MapMenuClear
//
this.MapMenuClear.Text = "Clear Map";
this.MapMenuClear.Click +=
    new System.EventHandler(this.MapMenuClear_Click);
//
// MapMenuMark
//
this.MapMenuMark.Enabled = false;
this.MapMenuMark.MenuItems.Add(this.MapMenuMarkCurPosit);
this.MapMenuMark.MenuItems.Add(this.MapMenuMarkCoord);
this.MapMenuMark.Text = "Mark...";
//
// MapMenuMarkCurPosit
//
this.MapMenuMarkCurPosit.Enabled = false;
this.MapMenuMarkCurPosit.Text = "Current Position";
```

```
//
// MapMenuMarkCoord
//
this.MapMenuMarkCoord.Enabled = false;
this.MapMenuMarkCoord.Text = "User Coordinates";
//
// MapMenuFind
//
this.MapMenuFind.Enabled = false;
this.MapMenuFind.MenuItems.Add(this.MapMenuFindCoord);
this.MapMenuFind.MenuItems.Add(this.MapMenuFindLastPosit);
this.MapMenuFind.MenuItems.Add(this.MapMenuFindCurrPosit);
this.MapMenuFind.Text = "Find...";
//
// MapMenuFindCoord
//
this.MapMenuFindCoord.Enabled = false;
this.MapMenuFindCoord.Text = "User Coordinates";
//
// MapMenuFindLastPosit
//
this.MapMenuFindLastPosit.Enabled = false;
this.MapMenuFindLastPosit.Text = "Last Position";
//
// MapMenuFindCurrPosit
//
this.MapMenuFindCurrPosit.Enabled = false;
this.MapMenuFindCurrPosit.Text = "Current Position";
//
// MapMenuTest
//
this.MapMenuTest.Text = "***Test***";
this.MapMenuTest.Click +=
    new System.EventHandler(this.MapMenuTest_Click);
//
// MapMenuNew
//
this.MapMenuNew.Text = "New Map";
this.MapMenuNew.Click +=
    new System.EventHandler(this.MapMenuNew_Click);
//
// MapMenuProperties
//
this.MapMenuProperties.Text = "Properties";
this.MapMenuProperties.Click +=
    new System.EventHandler(this.MapMenuProperties_Click);
//
// hBar
//
this.hBar.Location = new System.Drawing.Point(0, 256);
this.hBar.Maximum = 1168;
this.hBar.Size = new System.Drawing.Size(224, 16);
this.hBar.ValueChanged +=
    new System.EventHandler(this.hBar_ValueChanged);
//
// vBar
//
```

```
this.vBar.Location = new System.Drawing.Point(224, 0);
this.vBar.Maximum = 742;
this.vBar.Size = new System.Drawing.Size(16, 256);
this.vBar.ValueChanged +=
    new System.EventHandler(this.vBar_ValueChanged);
//
// FillerPnl
//
this.FillerPnl.Location =
    new System.Drawing.Point(224, 256);
this.FillerPnl.Size = new System.Drawing.Size(16, 16);
//
// viewPort
//
this.viewPort.Size = new System.Drawing.Size(1398, 995);
//
// curPosition
//
this.curPosition.Location =
    new System.Drawing.Point(88, 152);
this.curPosition.Size = new System.Drawing.Size(8, 8);
this.curPosition.Visible = false;
this.curPosition.Click +=
    new System.EventHandler(this.curPosition_Click);
//
// demoTimer
//
this.demoTimer.Interval = 1000;
this.demoTimer.Tick +=
    new System.EventHandler(this.demoTimer_Tick);
//
// GPSTimer
//
this.GPSTimer.Interval = 5000;
this.GPSTimer.Tick +=
    new System.EventHandler(this.GPSTimer_Tick);
//
// FileMenuLoad
//
this.FileMenuLoad.Enabled = false;
this.FileMenuLoad.Text = "Load Saved Log";
//
// FileMenuConnect
//
this.FileMenuConnect.Enabled = false;
this.FileMenuConnect.Text = "Connect to Server";
//
// FileMenuSend
//
this.FileMenuSend.Enabled = false;
this.FileMenuSend.MenuItems.Add(this.FileMenuSendFile);
this.FileMenuSend.MenuItems.Add(this.FileMenuSendQMsg);
this.FileMenuSend.MenuItems.Add(this.FileMenuSendMsg);
this.FileMenuSend.Text = "Send...";
//
// FileMenuSendFile
//
```

```
        this.FileMenuSendFile.Enabled = false;
        this.FileMenuSendFile.Text = "File";
        //
        // FileMenuSendMsg
        //
        this.FileMenuSendMsg.Enabled = false;
        this.FileMenuSendMsg.Text = "Message";
        //
        // FileMenuSendQMsg
        //
        this.FileMenuSendQMsg.Enabled = false;
        this.FileMenuSendQMsg.Text = "Quick Message";
        //
        // Form1
        //
        this.Controls.Add(this.hBar);
        this.Controls.Add(this.vBar);
        this.Controls.Add(this.curPosition);
        this.Controls.Add(this.FillerPnl);
        this.Controls.Add(this.viewPort);
        this.Menu = this.mainMenu1;
        this.Text = "NPS Navigator";

    }
    #endregion

    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    static void Main()
    {
        Application.Run(new Form1());
    }

    /// <summary>
    /// The main thread that manages the client side of
    /// communications and handles incoming messages in
    /// the appropriate manner.
    /// </summary>
    private void runClient()
    {
        try
        {
            //display client status to user
            MessageBox.Show("Attempting Connection...");

            //establish TCP/IP connection with Master Station on
            //  port 5000; IP address may change if DHCP is being
            //  used
            client = new TcpClient();
            client.Connect("172.20.146.158", 5000);

            //display client status to user
            MessageBox.Show("Setting up connection...");

            //establish TCP/IP resources
            stream = client.GetStream();
```

```
            writer = new BinaryWriter(stream);
            reader = new BinaryReader(stream);

            //display client status to user
            MessageBox.Show("Connection established.");

            do
            {
                try
                {
                    //read a packet in
                    srvMsg = reader.ReadString();
                    //Display packet contents for user
                    MessageBox.Show("FROM SERVER>> " + srvMsg);
                }
                //catch and display errors
                catch(Exception)
                {
                    MessageBox.Show(
                        "Error reading server data...");
                    break;
                }
            }while(srvMsg != "terminate");
        }
        //catch and display errors
        catch (Exception error)
        {
            MessageBox.Show("Error in connection loop: " +
                error.ToString());
        }
    }

    /// <summary>
    /// Exit the program
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void FileMenuExit_Click(object sender,
        System.EventArgs e)
    {
        Application.Exit();
    }

    /// <summary>
    /// Handles horizontal scrolling and updates veiwport then
    /// redraws position if necessary
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void hBar_ValueChanged(object sender,
        System.EventArgs e)
    {
        //if user is scrolling
        if(userchange)
        {
            //move left edge
            viewPort.Left = -hBar.Value;
```

```
            //show scrolling
            viewPort.Refresh();

            //move upperleft corner of viewport
            upperLeft = new Point(hBar.Value, upperLeft.Y);

            //if a position exists then draw it
            if (positExists)
            {
                this.drawPosition();
            }
        }
    }

    /// <summary>
    /// Handles Vertical scrolling and updates veiwport then
    /// redraws position if necessary
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void vBar_ValueChanged(object sender,
                                   System.EventArgs e)
    {
        //if user is scrolling
        if (userchange)
        {
            //move top edge
            viewPort.Top = -vBar.Value;

            //show scrolling
            viewPort.Refresh();

            //move upperleft corner of veiwport
            upperLeft = new Point(upperLeft.X, vBar.Value);

            //if a position exists then draw it
            if(positExists)
            {
                this.drawPosition();
            }
        }
    }

    /// <summary>
    /// Used in early stages of development to enter
    /// demonstration mode
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void GPSMenuDemo_Click(object sender,
                                   System.EventArgs e)
    {
        //start timer
        demoTimer.Enabled = true;
    }
```

```
/// <summary>
/// Draw user GPS positions on screen and sends the GPS data
/// back to the Master Station along with track tolerence
/// flag information
/// </summary>
private void drawPosition()
{
    //define x and y viewport parameters to NPS map image
    int xStart = upperLeft.X;
    int xStop = xStart + 230;
    int yStart = upperLeft.Y;
    int yStop = yStart + 285;

    //new upperleft position for calculating moved distances
    //  in pixels
    Point newUpperLeft;

    //x and y pixels moved from last position
    int xDiff;
    int yDiff;

    //used for autoscroll
    bool again = true;

    //defines the scrolling direction
    string hMove;
    string vMove;

    //if current GPS plot is veiwable based on current
    //  viewport location
    if (((xStart < GPSman.plotCurX()) &&
         (GPSman.plotCurX() < xStop)) &&
        ((yStart < GPSman.plotCurY()) &&
         (GPSman.plotCurY() < yStop)))
    {
        //set positional cursor location
        curPosition.Location = new Point(
            (GPSman.plotCurX()-upperLeft.X),
            (GPSman.plotCurY()-upperLeft.Y));
        //if within tolerence, color cursor green and draw it
        if(GPSman.isOnTrack())
        {
            curPosition.BackColor = Color.Green;
            curPosition.Visible = true;

            //Tell Master Station to paint posit green
            writer.Write("W");
        }
        //if not within tolerence, color cursor red and draw
        //  it
        else
        {
            curPosition.BackColor = Color.Red;
            curPosition.Visible = true;

            //Tell Master Station to paint posit red and to
            //  display warning message
```

```
            try
            {
                writer.Write("X");
                writer.Write("MSmall Boat has exceeded " +
                    "maximum distance from track (50 yds.)");
            }
            //catch and display errors
            catch(Exception)
            {
                MessageBox.Show("Error writing distance " +
                    "warning to Master Station.");
            }
        }
    }
    //used for auto scrolling feature
    else
    {
        //determine new point where upperLeft should be to
        //  center cursor
        newUpperLeft = new Point((GPSman.plotCurX() - 115),
                                 (GPSman.plotCurY() - 142));

        //determine direction of required horizontal scroll
        //  movement
        if (newUpperLeft.X > upperLeft.X)
        {
            xDiff = newUpperLeft.X - upperLeft.X;
            hMove = "right";
        }
        else
        {
            xDiff = upperLeft.X - newUpperLeft.X;
            hMove = "left";
        }

        //determine direction of required vertical scroll
        //  movement
        if (newUpperLeft.Y > upperLeft.Y)
        {
            yDiff = newUpperLeft.Y - upperLeft.Y;
            vMove = "down";
        }
        else
        {
            yDiff = upperLeft.Y - newUpperLeft.Y;
            vMove = "up";
        }

        //scroll viewport until current position is centered
        while(again)
        {
            //if the vertical scroll is not where it should
            //  be
            if(vBar.Value != newUpperLeft.Y)
            {
                //if scrolling up
                if (vMove == "up")
```

```
        {
            //mark as internal scroll and decrement
            //  scrollbar value
            userchange = false;
            vBar.Value--;

            //set left and top edges of viewport and
            //  upperleft
            viewPort.Left = -hBar.Value;
            upperLeft = new Point(hBar.Value,
                upperLeft.Y);
            viewPort.Top = -vBar.Value;
            upperLeft = new Point(upperLeft.X,
                vBar.Value);

        }
        //if scrolling down
        else
        {
            //mark as internal scroll and increment
            //  scrollbar value
            userchange = false;
            vBar.Value++;

            //set left and top edges of viewport and
            //  upperleft
            viewPort.Left = -hBar.Value;
            upperLeft = new Point(hBar.Value,
                upperLeft.Y);
            viewPort.Top = -vBar.Value;
            upperLeft = new Point(upperLeft.X,
                vBar.Value);
        }
    }

    //if the horizontal scroll is not where it
    //  should be
    if(hBar.Value != newUpperLeft.X)
    {
        //if scrolling right
        if (hMove == "right")
        {
            //mark as internal scroll and increment
            //  scrollbar value
            userchange = false;
            hBar.Value++;

            //set left and top edges of viewport and
            //  upperleft
            viewPort.Left = -hBar.Value;
            upperLeft = new Point(hBar.Value,
                upperLeft.Y);
            viewPort.Top = -vBar.Value;
            upperLeft = new Point(upperLeft.X,
                vBar.Value);
        }
        //if scrolling left
```

```
        else
        {
            //mark as internal scroll and decrement
            //  scrollbar value
            userchange = false;
            hBar.Value--;

            //set left and top edges of viewport and
            //  upperleft
            viewPort.Left = -hBar.Value;
            upperLeft = new Point(hBar.Value,
                upperLeft.Y);
            viewPort.Top = -vBar.Value;
            upperLeft = new Point(upperLeft.X,
                vBar.Value);
        }
    }

    //refresh the viewport with new values
    viewPort.Refresh();

    //is viewport correctly positioned?
    if ((hBar.Value == newUpperLeft.X) &&
        (vBar.Value == newUpperLeft.Y))
    {
        //stop autoscroll
        again = false;
        //default back to user initiated changes
        userchange = true;
    }

    //set positional cursor location
    curPosition.Location = new Point(
        (GPSman.plotCurX()-upperLeft.X),
        (GPSman.plotCurY()-upperLeft.Y));

    //if within tolerence color cursor green and
    //  draw it
    if(GPSman.isOnTrack())
    {
        curPosition.BackColor = Color.Green;
        curPosition.Visible = true;

        //Tell Master Station that small boat to
        //  paint posit green
        writer.Write("W");
    }
    //if not within tolerence color cursor red and
    //  draw it
    else
    {
        curPosition.BackColor = Color.Red;
        curPosition.Visible = true;

        //send warning to Master Station along with
        //  cursor color flag
        try
```

```
                {
                    writer.Write("MSmall Boat has exceeded" +
                        " maximum distance from track " +
                        "(50 yds.)");
                    writer.Write("X");
                }
                //catch and display errors
                catch(Exception)
                {
                    MessageBox.Show("Error writing distance" +
                        "warning to Master Station.");
                }
            }
        }


    }
}

/// <summary>
/// This is a menu option used for testing purposes
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void MapMenuTest_Click(object sender,
    System.EventArgs e)
{
    //Creates an instance of FormMapInfo
    this.refFormMapDetailsTest =
        new FormMapDetailsTest(GPSman);

    //Shows the form
    this.refFormMapDetailsTest.Show();
}

/// <summary>
/// Timer action for Demonstration mode
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void demoTimer_Tick(object sender,
    System.EventArgs e)
{
    //there is now a posit to draw
    positExists = true;

    //EOF
    if (gpsLine == null)
    {
        //stop timer
        demoTimer.Enabled = false;
        //alert exiting from demo mode
        MessageBox.Show("End of Demo");
    }
    //not EOF
    else
    {
```

```
        //parse the gpsLine and draw the position
        GPSman.NMEAParser(gpsLine);
        this.drawPosition();
    }

}

/// <summary>
/// clear positon marker from the map
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void MapMenuClear_Click(object sender,
    System.EventArgs e)
{
    curPosition.Visible = false;
}

/// <summary>
/// Gives positional information (time, lat, long) when the
/// cursor is clicked
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void curPosition_Click(object sender,
    System.EventArgs e)
{
    //read string and set delimiters for parsing
    string str = GPSman.getFixDetails();
    int latDelimiter = str.IndexOf("N",0,str.Length);
    int longStart = latDelimiter + 1;
    int latLength = latDelimiter - 8;
    int longLength = str.Length - latDelimiter;
    int timeDelimiter = str.IndexOf(":",0,5);
    timeDelimiter = timeDelimiter + 3;

    if(GPSman.isRunning())
    {
        MessageBox.Show("Fix Time:  " + str.Substring(0,
            timeDelimiter) + "\n" + "Latitude:  " +
            str.Substring(timeDelimiter,latLength + 1) +
            "\n" + "Longitude: " + str.Substring(longStart,
            longLength) + "\n", "Fix Details...");
    }
    else
    {
        MessageBox.Show("Fix Time:  " + str.Substring(0,8) +
            "\n" + "Latitude:  " + str.Substring(8,
            latLength + 1) + "\n" + "Longitude: " +
            str.Substring(longStart,longLength) + "\n",
            "Fix Details...");
    }
}

/// <summary>
/// Menu option used to connect to the GPS receiver for data
/// retreival based on the status of menu option's text
```

```
/// display.
/// </summary>
private void GPSMenuConnection_Click(object sender,
    System.EventArgs e)
{
    //user is trying to connect
    if (GPSMenuConnection.Text == "Connect")
    {
        //connect to GPS receiver
        GPSman.start();
        //change text display
        GPSMenuConnection.Text = "Disconnect";
        //start GPSTimer
        GPSTimer.Enabled = true;
    }
    //user is trying to disconnect
    else
    {
        //disconnect from GPS receiver
        GPSman.stop();
        //change text display
        GPSMenuConnection.Text = "Connect";
        //stop GPSTimer
        GPSTimer.Enabled = false;
    }
}

/// <summary>
/// GPSTimer event handler that manages a complete cycle of
/// timer action.  Takes place every five seconds.
/// </summary>
private void GPSTimer_Tick(object sender, System.EventArgs e)
{
    //sets ne posit flag for potential drawPosition call
    positExists = true;

    //if GPS receiver connection is active and synchronized
    //  with constellation
    if (GPSman.isRunning())
    {
        //posit is available and located within map
        //  parameters
        if(GPSman.getLiveData())
        {
            try
            {
                //send position to Master Station
                writer.Write(GPSman.sendDataStr);
            }
            //catch and display errors
            catch(Exception)
            {
                MessageBox.Show("Error writing packet: " +
                    GPSman.sendDataStr);
            }

            //draw position on screen
```

```
            this.drawPosition();
        }
    }
    //connection is not active or not synchronized with
    //  constellation
    else
    {
        MessageBox.Show("Waiting for satellite " +
            "synchronization.");
    }
}


/// <summary>
/// Menu option that enables user to load a new map and its
/// related parameters
/// </summary>
private void MapMenuNew_Click(object sender,
    System.EventArgs e)
{
    //provide OpenFileDialog box for user to pick map
    OpenFileDialog openFileDialog1 = new OpenFileDialog();
    openFileDialog1.InitialDirectory = "\\My Pictures\\" ;
    openFileDialog1.Filter =
        "txt files (*.txt)|*.txt|All files (*.*)|*.*" ;
    openFileDialog1.FilterIndex = 2 ;

    //user selected a map
    if(openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        //display map in viewPort
        viewPort.Image = new System.Drawing.Bitmap(
            openFileDialog1.FileName);

        //Creates an instance of FormMapInfo to begin
        //  loading parameters
        this.refFormMapInfoLat = new FormMapInfoLat(GPSman);

        //Shows the form
        this.refFormMapInfoLat.Show();
    }
}

/// <summary>
/// Displays currently loaded map parameters for user review
/// </summary>
private void MapMenuProperties_Click(object sender,
    System.EventArgs e)
{
    this.refFormMapConfirmation =
        new FormMapConfirmation(GPSman);
    this.refFormMapConfirmation.Show();
}

/// <summary>
/// Reads waypoints in from external file and makes them
/// ready for use
```

```
/// </summary>
private void getWaypoints()
{
    string waypoint;
    int idx = 0;
    decimal wpLat;
    decimal wpLong;
    decimal deg;
    decimal min;
    decimal sec;

    //read first waypoint from waypoints.txt
    waypoint = sr.ReadLine();

    //not EOF
    while(waypoint != null)
    {
        //load waypoint lat and long
        GPSman.mapper.wayptLat[idx] =
            waypoint.Substring(0,9);
        GPSman.mapper.wayptLong[idx] =
            waypoint.Substring(9,10);

        //increment waypoint index
        idx++;
        //get next waypoint
        waypoint = sr.ReadLine();
    }

    //close file reader
    sr.Close();

    //convert waypoint strings to decimal data types
    for(int i = 0; i < idx; i++)
    {
        //Build degree decimal format for wpLat
        sec = (decimal.Parse(
            GPSman.mapper.wayptLat[i].Substring(4,5)) / 60);
        min = (decimal.Parse(
            GPSman.mapper.wayptLat[i].Substring(2,2)) + sec)
            / 60;
        deg = (decimal.Parse(
            GPSman.mapper.wayptLat[i].Substring(0,2))) + min;
        wpLat = deg;

        //Build degree decimal format for wpLong
        sec = (decimal.Parse(
            GPSman.mapper.wayptLong[i].Substring(5,5)) / 60);
        min = (decimal.Parse(
            GPSman.mapper.wayptLong[i].Substring(3,2)) + sec)
            / 60;
        deg = (decimal.Parse(
            GPSman.mapper.wayptLong[i].Substring(0,3))) + min;
        wpLong = deg;

        //store decimal formatted lat and long for waypoints
        GPSman.mapper.wpLatDecDeg[i] = wpLat;
```

```
                GPSman.mapper.wpLongDecDeg[i] = wpLong;
            }
        }

    }
}
```

2. FormMapConfirmation.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace MobileDeNS
{
    /// <summary>
    /// FormMapConfirmation is  helper form that allows the user to
    /// review currently loaded map parameters in a read only format.
    /// It is the final form displayed in the map parameter loading
    /// procedure, and can be accessed directly through the Map menu
    /// found in MobileDeNS Form1.
    /// </summary>
    public class FormMapConfirmation : System.Windows.Forms.Form
    {
        private GPSHandler GPS;
        private System.Windows.Forms.Form refFormMapInfoLat;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.Button ChangeBtn;
        private System.Windows.Forms.Button OKBtn;
        private System.Windows.Forms.Label upperLat;
        private System.Windows.Forms.Label leftLong;
        private System.Windows.Forms.Label rightLong;
        private System.Windows.Forms.Label pixelSize;
        private System.Windows.Forms.Label lowerLat;
        private System.Windows.Forms.Label label1;

        /// <summary>
        /// FormMapConfirmation constructor.  In requires an
        /// instance of GPSHandler be passed as a parameter.
        /// </summary>
        public FormMapConfirmation(GPSHandler inGPS)
        {
            //Build the form
            InitializeComponent();

            //set pointer to GPSHandler argument
            GPS = inGPS;

            //read relevent data from GPSHandler
            upperLat.Text = GPS.mapper.upperLat.ToString() + " " +
                GPS.mapper.upperNSHem;
            lowerLat.Text = GPS.mapper.lowerLat.ToString() + " " +
```

```
            GPS.mapper.lowerNSHem;
        leftLong.Text = GPS.mapper.leftLong.ToString() + " " +
            GPS.mapper.leftEWHem;
        rightLong.Text = GPS.mapper.rightLong.ToString() + " " +
            GPS.mapper.rightEWHem;
        pixelSize.Text = GPS.mapper.pixelHeight + " x " +
            GPS.mapper.pixelWidth;
    }

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    protected override void Dispose( bool disposing )
    {
        base.Dispose( disposing );
    }

    #region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        this.label1 = new System.Windows.Forms.Label();
        this.label2 = new System.Windows.Forms.Label();
        this.label3 = new System.Windows.Forms.Label();
        this.label4 = new System.Windows.Forms.Label();
        this.label5 = new System.Windows.Forms.Label();
        this.upperLat = new System.Windows.Forms.Label();
        this.leftLong = new System.Windows.Forms.Label();
        this.rightLong = new System.Windows.Forms.Label();
        this.pixelSize = new System.Windows.Forms.Label();
        this.lowerLat = new System.Windows.Forms.Label();
        this.ChangeBtn = new System.Windows.Forms.Button();
        this.OKBtn = new System.Windows.Forms.Button();
        //
        // label1
        //
        this.label1.Location = new System.Drawing.Point(8, 16);
        this.label1.Size = new System.Drawing.Size(224, 16);
        this.label1.Text = "Upper Latitude:";
        //
        // label2
        //
        this.label2.Location = new System.Drawing.Point(8, 56);
        this.label2.Size = new System.Drawing.Size(224, 16);
        this.label2.Text = "Lower Latitude:";
        //
        // label3
        //
        this.label3.Location = new System.Drawing.Point(8, 96);
        this.label3.Size = new System.Drawing.Size(224, 16);
        this.label3.Text = "Left Longitude:";
        //
        // label4
        //
```

```
this.label4.Location = new System.Drawing.Point(8, 136);
this.label4.Size = new System.Drawing.Size(224, 16);
this.label4.Text = "Right Longitude:";
//
// label5
//
this.label5.Location = new System.Drawing.Point(8, 176);
this.label5.Size = new System.Drawing.Size(224, 16);
this.label5.Text = "Pixel Size (H x W):";
//
// upperLat
//
this.upperLat.ForeColor =
    System.Drawing.SystemColors.ControlText;
this.upperLat.Location = new System.Drawing.Point(16, 32);
this.upperLat.Size = new System.Drawing.Size(208, 16);
//
// leftLong
//
this.leftLong.Location =
    new System.Drawing.Point(16, 112);
this.leftLong.Size = new System.Drawing.Size(208, 16);
//
// rightLong
//
this.rightLong.Location =
    new System.Drawing.Point(16, 152);
this.rightLong.Size = new System.Drawing.Size(208, 16);
//
// pixelSize
//
this.pixelSize.Location =
    new System.Drawing.Point(16, 192);
this.pixelSize.Size = new System.Drawing.Size(208, 16);
//
// lowerLat
//
this.lowerLat.Location =
    new System.Drawing.Point(16, 72);
this.lowerLat.Size = new System.Drawing.Size(208, 16);
//
// ChangeBtn
//
this.ChangeBtn.Location =
    new System.Drawing.Point(168, 224);
this.ChangeBtn.Size = new System.Drawing.Size(56, 24);
this.ChangeBtn.Text = "Change";
this.ChangeBtn.Click +=
    new System.EventHandler(this.ChangeBtn_Click);
//
// OKBtn
//
this.OKBtn.Location = new System.Drawing.Point(128, 224);
this.OKBtn.Size = new System.Drawing.Size(32, 24);
this.OKBtn.Text = "OK";
this.OKBtn.Click +=
    new System.EventHandler(this.OKBtn_Click);
```

```
            //
            // FormMapConfirmation
            //
            this.Controls.Add(this.OKBtn);
            this.Controls.Add(this.ChangeBtn);
            this.Controls.Add(this.lowerLat);
            this.Controls.Add(this.pixelSize);
            this.Controls.Add(this.rightLong);
            this.Controls.Add(this.leftLong);
            this.Controls.Add(this.upperLat);
            this.Controls.Add(this.label5);
            this.Controls.Add(this.label4);
            this.Controls.Add(this.label3);
            this.Controls.Add(this.label2);
            this.Controls.Add(this.label1);
            this.Text = "You have entered:";

        }
        #endregion

        /// <summary>
        /// Signifies that user has reviewed all data; closes form.
        /// </summary>
        private void OKBtn_Click(object sender, System.EventArgs e)
        {
            this.Dispose();
            this.Close();
        }

        /// <summary>
        /// Signifies that user needs to change current parameters
        /// and loads the form that begins the parameter
        /// specification process.
        /// </summary>
        private void ChangeBtn_Click(object sender,
            System.EventArgs e)
        {
            //Creates an instance of FormMapInfo
            this.refFormMapInfoLat = new FormMapInfoLat(GPS);

            //Shows the form
            this.refFormMapInfoLat.Show();

            this.Dispose();
            this.Close();
        }

    }
}
```

3. FormMapDetailsTest.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
```

```
namespace MobileDeNS
{
    /// <summary>
    /// FormMapConfirmation is  helper form that allows the user to
    /// review currently loaded map parameters in a read only format.
    /// It is the final form displayed in the map parameter loading
    /// procedure, and can be accessed directly through the Map menu
    /// found in MobileDeNS Form1.
    /// </summary>
    public class FormMapConfirmation : System.Windows.Forms.Form
    {
        private GPSHandler GPS;
        private System.Windows.Forms.Form refFormMapInfoLat;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.Button ChangeBtn;
        private System.Windows.Forms.Button OKBtn;
        private System.Windows.Forms.Label upperLat;
        private System.Windows.Forms.Label leftLong;
        private System.Windows.Forms.Label rightLong;
        private System.Windows.Forms.Label pixelSize;
        private System.Windows.Forms.Label lowerLat;
        private System.Windows.Forms.Label label1;

        /// <summary>
        /// FormMapConfirmation constructor.  In requires an
        /// instance of GPSHandler be passed as a parameter.
        /// </summary>
        public FormMapConfirmation(GPSHandler inGPS)
        {
            //Build the form
            InitializeComponent();

            //set pointer to GPSHandler argument
            GPS = inGPS;

            //read relevent data from GPSHandler
            upperLat.Text = GPS.mapper.upperLat.ToString() + " " +
                GPS.mapper.upperNSHem;
            lowerLat.Text = GPS.mapper.lowerLat.ToString() + " " +
                GPS.mapper.lowerNSHem;
            leftLong.Text = GPS.mapper.leftLong.ToString() + " " +
                GPS.mapper.leftEWHem;
            rightLong.Text = GPS.mapper.rightLong.ToString() + " " +
                GPS.mapper.rightEWHem;
            pixelSize.Text = GPS.mapper.pixelHeight + " x " +
                GPS.mapper.pixelWidth;
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
```

```
        base.Dispose( disposing );
    }

    #region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        this.label1 = new System.Windows.Forms.Label();
        this.label2 = new System.Windows.Forms.Label();
        this.label3 = new System.Windows.Forms.Label();
        this.label4 = new System.Windows.Forms.Label();
        this.label5 = new System.Windows.Forms.Label();
        this.upperLat = new System.Windows.Forms.Label();
        this.leftLong = new System.Windows.Forms.Label();
        this.rightLong = new System.Windows.Forms.Label();
        this.pixelSize = new System.Windows.Forms.Label();
        this.lowerLat = new System.Windows.Forms.Label();
        this.ChangeBtn = new System.Windows.Forms.Button();
        this.OKBtn = new System.Windows.Forms.Button();
        //
        // label1
        //
        this.label1.Location = new System.Drawing.Point(8, 16);
        this.label1.Size = new System.Drawing.Size(224, 16);
        this.label1.Text = "Upper Latitude:";
        //
        // label2
        //
        this.label2.Location = new System.Drawing.Point(8, 56);
        this.label2.Size = new System.Drawing.Size(224, 16);
        this.label2.Text = "Lower Latitude:";
        //
        // label3
        //
        this.label3.Location = new System.Drawing.Point(8, 96);
        this.label3.Size = new System.Drawing.Size(224, 16);
        this.label3.Text = "Left Longitude:";
        //
        // label4
        //
        this.label4.Location = new System.Drawing.Point(8, 136);
        this.label4.Size = new System.Drawing.Size(224, 16);
        this.label4.Text = "Right Longitude:";
        //
        // label5
        //
        this.label5.Location = new System.Drawing.Point(8, 176);
        this.label5.Size = new System.Drawing.Size(224, 16);
        this.label5.Text = "Pixel Size (H x W):";
        //
        // upperLat
        //
        this.upperLat.ForeColor =
            System.Drawing.SystemColors.ControlText;
```

```
this.upperLat.Location = new System.Drawing.Point(16, 32);
this.upperLat.Size = new System.Drawing.Size(208, 16);
//
// leftLong
//
this.leftLong.Location =
    new System.Drawing.Point(16, 112);
this.leftLong.Size = new System.Drawing.Size(208, 16);
//
// rightLong
//
this.rightLong.Location =
    new System.Drawing.Point(16, 152);
this.rightLong.Size = new System.Drawing.Size(208, 16);
//
// pixelSize
//
this.pixelSize.Location =
    new System.Drawing.Point(16, 192);
this.pixelSize.Size = new System.Drawing.Size(208, 16);
//
// lowerLat
//
this.lowerLat.Location =
    new System.Drawing.Point(16, 72);
this.lowerLat.Size = new System.Drawing.Size(208, 16);
//
// ChangeBtn
//
this.ChangeBtn.Location =
    new System.Drawing.Point(168, 224);
this.ChangeBtn.Size = new System.Drawing.Size(56, 24);
this.ChangeBtn.Text = "Change";
this.ChangeBtn.Click +=
    new System.EventHandler(this.ChangeBtn_Click);
//
// OKBtn
//
this.OKBtn.Location = new System.Drawing.Point(128, 224);
this.OKBtn.Size = new System.Drawing.Size(32, 24);
this.OKBtn.Text = "OK";
this.OKBtn.Click +=
    new System.EventHandler(this.OKBtn_Click);
//
// FormMapConfirmation
//
this.Controls.Add(this.OKBtn);
this.Controls.Add(this.ChangeBtn);
this.Controls.Add(this.lowerLat);
this.Controls.Add(this.pixelSize);
this.Controls.Add(this.rightLong);
this.Controls.Add(this.leftLong);
this.Controls.Add(this.upperLat);
this.Controls.Add(this.label5);
this.Controls.Add(this.label4);
this.Controls.Add(this.label3);
this.Controls.Add(this.label2);
```

```
            this.Controls.Add(this.label1);
            this.Text = "You have entered:";

        }
        #endregion

        /// <summary>
        /// Signifies that user has reviewed all data; closes form.
        /// </summary>
        private void OKBtn_Click(object sender, System.EventArgs e)
        {
            this.Dispose();
            this.Close();
        }

        /// <summary>
        /// Signifies that user needs to change current parameters
        /// and loads the form that begins the parameter
        /// specification process.
        /// </summary>
        private void ChangeBtn_Click(object sender,
            System.EventArgs e)
        {
            //Creates an instance of FormMapInfo
            this.refFormMapInfoLat = new FormMapInfoLat(GPS);

            //Shows the form
            this.refFormMapInfoLat.Show();

            this.Dispose();
            this.Close();
        }

    }
}
```

4. FormMapInfoLat.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace MobileDeNS
{
    /// <summary>
    /// FormMapInfoLat is a helper form that allows the user to
    /// review currently loaded latitude values in a read/write
    /// format.  Hemipsphere indicators are provided, though they
    /// are not used in the prototype version of this program.  They
    /// are included for future versions that will use these values.
    /// It can be accessed through the Map menu found in MobileDeNS
    /// Form1 by selecting New Map.  This is the first form in the
    /// map parameter specification process.
    /// </summary>
    public class FormMapInfoLat : System.Windows.Forms.Form
```

```
{
    private GPSHandler GPS;
    private System.Windows.Forms.Form refFormMapInfoLong;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.Label label2;
    private System.Windows.Forms.TextBox upperLatDeg;
    private System.Windows.Forms.TextBox upperLatMin;
    private System.Windows.Forms.TextBox upperLatSec;
    private System.Windows.Forms.TextBox lowerLatSec;
    private System.Windows.Forms.TextBox lowerLatMin;
    private System.Windows.Forms.TextBox lowerLatDeg;
    private System.Windows.Forms.ComboBox upperLatHem;
    private System.Windows.Forms.ComboBox lowerLatHem;
    private System.Windows.Forms.Button okBtn;
    private System.Windows.Forms.Button cancelBtn;
    private Microsoft.WindowsCE.Forms.InputPanel inputPanel1;

    /// <summary>
    /// FormMapInfoLat constructor.  Requires GPSHandler to be
    /// passed as an argument.
    /// </summary>
    public FormMapInfoLat(GPSHandler inGPS)
    {
        //Build the form
        InitializeComponent();

        //build the drop down menu for hemispheres
        upperLatHem.Items.Add("N");
        upperLatHem.Items.Add("S");
        lowerLatHem.Items.Add("N");
        lowerLatHem.Items.Add("S");

        //set pointer to GPSHandler argument
        GPS = inGPS;
    }

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    protected override void Dispose( bool disposing )
    {
        base.Dispose( disposing );
    }

    #region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        this.upperLatDeg = new System.Windows.Forms.TextBox();
        this.upperLatMin = new System.Windows.Forms.TextBox();
        this.upperLatSec = new System.Windows.Forms.TextBox();
        this.lowerLatSec = new System.Windows.Forms.TextBox();
        this.lowerLatMin = new System.Windows.Forms.TextBox();
        this.lowerLatDeg = new System.Windows.Forms.TextBox();
```

```
this.upperLatHem = new System.Windows.Forms.ComboBox();
this.lowerLatHem = new System.Windows.Forms.ComboBox();
this.label1 = new System.Windows.Forms.Label();
this.label2 = new System.Windows.Forms.Label();
this.okBtn = new System.Windows.Forms.Button();
this.cancelBtn = new System.Windows.Forms.Button();
//
// upperLatDeg
//
this.upperLatDeg.Location =
    new System.Drawing.Point(16, 32);
this.upperLatDeg.Size = new System.Drawing.Size(32, 22);
this.upperLatDeg.Text = "36";
//
// upperLatMin
//
this.upperLatMin.Location =
    new System.Drawing.Point(72, 32);
this.upperLatMin.Size = new System.Drawing.Size(32, 22);
this.upperLatMin.Text = "36";
//
// upperLatSec
//
this.upperLatSec.Location =
    new System.Drawing.Point(120, 32);
this.upperLatSec.Size = new System.Drawing.Size(40, 22);
this.upperLatSec.Text = "57.76";
//
// lowerLatSec
//
this.lowerLatSec.Location =
    new System.Drawing.Point(120, 80);
this.lowerLatSec.Size = new System.Drawing.Size(40, 22);
this.lowerLatSec.Text = "31.56";
//
// lowerLatMin
//
this.lowerLatMin.Location =
    new System.Drawing.Point(72, 80);
this.lowerLatMin.Size = new System.Drawing.Size(32, 22);
this.lowerLatMin.Text = "34";
//
// lowerLatDeg
//
this.lowerLatDeg.Location =
    new System.Drawing.Point(16, 80);
this.lowerLatDeg.Size = new System.Drawing.Size(32, 22);
this.lowerLatDeg.Text = "36";
//
// upperLatHem
//
this.upperLatHem.Location =
    new System.Drawing.Point(184, 32);
this.upperLatHem.Size = new System.Drawing.Size(48, 22);
//
// lowerLatHem
//
```

```
        this.lowerLatHem.Location =
            new System.Drawing.Point(184, 80);
        this.lowerLatHem.Size = new System.Drawing.Size(48, 22);
        //
        // label1
        //
        this.label1.Location = new System.Drawing.Point(8, 16);
        this.label1.Size = new System.Drawing.Size(168, 16);
        this.label1.Text = "Upper latitude";
        //
        // label2
        //
        this.label2.Location = new System.Drawing.Point(8, 64);
        this.label2.Size = new System.Drawing.Size(168, 16);
        this.label2.Text = "Lower latitude";
        //
        // okBtn
        //
        this.okBtn.Location = new System.Drawing.Point(136, 128);
        this.okBtn.Size = new System.Drawing.Size(32, 24);
        this.okBtn.Text = "OK";
        this.okBtn.Click +=
            new System.EventHandler(this.okBtn_Click);
        //
        // cancelBtn
        //
        this.cancelBtn.Location =
            new System.Drawing.Point(176, 128);
        this.cancelBtn.Size = new System.Drawing.Size(56, 24);
        this.cancelBtn.Text = "Cancel";
        this.cancelBtn.Click +=
            new System.EventHandler(this.cancelBtn_Click);
        //
        // FormMapInfoLat
        //
        this.ClientSize = new System.Drawing.Size(240, 272);
        this.Controls.Add(this.cancelBtn);
        this.Controls.Add(this.okBtn);
        this.Controls.Add(this.label2);
        this.Controls.Add(this.label1);
        this.Controls.Add(this.lowerLatHem);
        this.Controls.Add(this.upperLatHem);
        this.Controls.Add(this.lowerLatSec);
        this.Controls.Add(this.lowerLatMin);
        this.Controls.Add(this.lowerLatDeg);
        this.Controls.Add(this.upperLatSec);
        this.Controls.Add(this.upperLatMin);
        this.Controls.Add(this.upperLatDeg);
        this.Text = "Map Information - Lat";

    }
    #endregion

    /// <summary>
    /// This button writes entered data into GPSHandler through
    /// the pointer.  It then opens the next form for longitude
    /// and closes itself.
```

```
        /// </summary>
        private void okBtn_Click(object sender, System.EventArgs e)
        {
            float latFloat;

            //converts upper minutes and seconds to float format
            //  representing decimal degrees.  Degrees are not used
            //  in the prototype, but are provided for future
            //  versions which will require them.
            latFloat = ((Convert.ToSingle(upperLatSec.Text)) / 60);
            latFloat = ((Convert.ToSingle(upperLatMin.Text)) +
                latFloat);
            latFloat = (Convert.ToSingle(upperLatDeg.Text) * 100) +
                latFloat;

            //write values to GPSHandler
            GPS.mapper.upperLat = latFloat;
            GPS.mapper.upperNSHem = upperLatHem.Text;

            //converts lower minutes and seconds to float format
            //  representing decimal degrees.  Degrees are not used
            //  in the prototype, but are provided for future
            //  versions which will require them.
            latFloat = ((Convert.ToSingle(lowerLatSec.Text)) / 60);
            latFloat = ((Convert.ToSingle(lowerLatMin.Text)) +
                latFloat);
            latFloat = (Convert.ToSingle(lowerLatDeg.Text) * 100) +
                latFloat;

            //write values to GPSHandler
            GPS.mapper.lowerLat = latFloat;
            GPS.mapper.lowerNSHem = lowerLatHem.Text;

            //Creates an instance of FormMapInfo
            this.refFormMapInfoLong = new FormMapInfoLong(GPS);

            //Shows the form
            this.refFormMapInfoLong.Show();

            this.Close();
        }

        /// <summary>
        /// Allows the user to cancel this process before values are
        /// written.
        /// </summary>
        private void cancelBtn_Click(object sender,
            System.EventArgs e)
        {
            this.Close();
        }

    }
}
```

5. FormMapInfoLong.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace MobileDeNS
{
    /// <summary>
    /// FormMapInfoLong is a helper form that allows the user to
    /// review currently loaded longitude values in a read/write
    /// format.  Hemipsphere indicators are provided, though they
    /// are not used in the prototype version of this program.  They
    /// are included for future versions that will use these values.
    /// It can be accessed through the Map menu found in MobileDeNS
    /// Form1 by selecting New Map, after entering data into the
    /// FormMapInfoLat page and selecting OK.  This is the second
    /// form in the map parameter specification process.
    /// </summary>
    public class FormMapInfoLong : System.Windows.Forms.Form
    {
        private GPSHandler GPS;
        private System.Windows.Forms.Form refFormMapInfoPix;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.ComboBox lLongHem;
        private System.Windows.Forms.TextBox lLongSec;
        private System.Windows.Forms.TextBox lLongMin;
        private System.Windows.Forms.TextBox lLongDeg;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.ComboBox rLongHem;
        private System.Windows.Forms.TextBox rLongSec;
        private System.Windows.Forms.TextBox rLongMin;
        private System.Windows.Forms.Button cancelBtn;
        private System.Windows.Forms.Button okBtn;
        private System.Windows.Forms.TextBox rLongDeg;

        /// <summary>
        /// FormMapInfoLong constructor.  Requires GPSHandler to be
        /// passed as an argument.
        /// </summary>
        public FormMapInfoLong(GPSHandler inGPS)
        {
            //build the form
            InitializeComponent();

            //build the drop down selection items
            rLongHem.Items.Add("E");
            rLongHem.Items.Add("W");
            lLongHem.Items.Add("E");
            lLongHem.Items.Add("W");

            //set pointer to GPSHandler argument
            GPS = inGPS;
        }
```

```
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
    base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.label4 = new System.Windows.Forms.Label();
    this.lLongHem = new System.Windows.Forms.ComboBox();
    this.lLongSec = new System.Windows.Forms.TextBox();
    this.lLongMin = new System.Windows.Forms.TextBox();
    this.lLongDeg = new System.Windows.Forms.TextBox();
    this.label3 = new System.Windows.Forms.Label();
    this.rLongHem = new System.Windows.Forms.ComboBox();
    this.rLongSec = new System.Windows.Forms.TextBox();
    this.rLongMin = new System.Windows.Forms.TextBox();
    this.rLongDeg = new System.Windows.Forms.TextBox();
    this.cancelBtn = new System.Windows.Forms.Button();
    this.okBtn = new System.Windows.Forms.Button();
    //
    // label4
    //
    this.label4.Location = new System.Drawing.Point(8, 16);
    this.label4.Size = new System.Drawing.Size(168, 16);
    this.label4.Text = "Left Longtitude";
    //
    // lLongHem
    //
    this.lLongHem.Location = new System.Drawing.Point(184, 32);
    this.lLongHem.Size = new System.Drawing.Size(48, 22);
    //
    // lLongSec
    //
    this.lLongSec.Location = new System.Drawing.Point(120, 32);
    this.lLongSec.Size = new System.Drawing.Size(40, 22);
    this.lLongSec.Text = "10.00";
    //
    // lLongMin
    //
    this.lLongMin.Location = new System.Drawing.Point(72, 32);
    this.lLongMin.Size = new System.Drawing.Size(32, 22);
    this.lLongMin.Text = "54";
    //
    // lLongDeg
    //
    this.lLongDeg.Location = new System.Drawing.Point(16, 32);
    this.lLongDeg.Size = new System.Drawing.Size(32, 22);
    this.lLongDeg.Text = "121";
    //
```

```
// label3
//
this.label3.Location = new System.Drawing.Point(8, 64);
this.label3.Size = new System.Drawing.Size(168, 16);
this.label3.Text = "Right Longitude";
//
// rLongHem
//
this.rLongHem.Location = new System.Drawing.Point(184, 80);
this.rLongHem.Size = new System.Drawing.Size(48, 22);
//
// rLongSec
//
this.rLongSec.Location = new System.Drawing.Point(120, 80);
this.rLongSec.Size = new System.Drawing.Size(40, 22);
this.rLongSec.Text = "57.94";
//
// rLongMin
//
this.rLongMin.Location = new System.Drawing.Point(72, 80);
this.rLongMin.Size = new System.Drawing.Size(32, 22);
this.rLongMin.Text = "50";
//
// rLongDeg
//
this.rLongDeg.Location = new System.Drawing.Point(16, 80);
this.rLongDeg.Size = new System.Drawing.Size(32, 22);
this.rLongDeg.Text = "121";
//
// cancelBtn
//
this.cancelBtn.Location =
    new System.Drawing.Point(176,128);
this.cancelBtn.Size = new System.Drawing.Size(56, 24);
this.cancelBtn.Text = "Cancel";
this.cancelBtn.Click +=
    new System.EventHandler(this.cancelBtn_Click);
//
// okBtn
//
this.okBtn.Location = new System.Drawing.Point(136, 128);
this.okBtn.Size = new System.Drawing.Size(32, 24);
this.okBtn.Text = "OK";
this.okBtn.Click +=
    new System.EventHandler(this.okBtn_Click);
//
// FormMapInfoLong
//
this.Controls.Add(this.cancelBtn);
this.Controls.Add(this.okBtn);
this.Controls.Add(this.label4);
this.Controls.Add(this.lLongHem);
this.Controls.Add(this.lLongSec);
this.Controls.Add(this.lLongMin);
this.Controls.Add(this.lLongDeg);
this.Controls.Add(this.label3);
this.Controls.Add(this.rLongHem);
```

```
        this.Controls.Add(this.rLongSec);
        this.Controls.Add(this.rLongMin);
        this.Controls.Add(this.rLongDeg);
        this.Text = "Map Information - Long";

    }
    #endregion

    /// <summary>
    /// This button writes entered data into GPSHandler through
    /// the pointer.  It then opens the next form for pixel info
    /// and closes itself.
    /// </summary>
    private void okBtn_Click(object sender, System.EventArgs e)
    {
        float longFloat;

        //converts left minutes and seconds to float format
        //  representing decimal degrees.  Degrees are not used
        //  in the prototype, but are provided for future
        //  versions which will require them.
        longFloat = ((Convert.ToSingle(lLongSec.Text)) / 60);
        longFloat = ((Convert.ToSingle(lLongMin.Text)) +
            longFloat);
        longFloat = (Convert.ToSingle(lLongDeg.Text) * 100) +
            longFloat;

        //write values to GPSHandler
        GPS.mapper.leftLong = longFloat;
        GPS.mapper.leftEWHem = lLongHem.Text;

        //converts right minutes and seconds to float format
        //  representing decimal degrees.  Degrees are not used
        //  in the prototype, but are provided for future
        //  versions which will require them.
        longFloat = ((Convert.ToSingle(rLongSec.Text)) / 60);
        longFloat = ((Convert.ToSingle(rLongMin.Text)) +
            longFloat);
        longFloat = (Convert.ToSingle(rLongDeg.Text) * 100) +
            longFloat;

        //write values to GPSHandler
        GPS.mapper.rightLong = longFloat;
        GPS.mapper.leftEWHem = rLongHem.Text;

        //Creates an instance of FormMapInfo
        this.refFormMapInfoPix = new FormMapInfoPix(GPS);

        //Shows the form
        this.refFormMapInfoPix.Show();

        this.Close();
    }

    /// <summary>
    /// Allows the user to cancel this process before values are
    /// written.
```

```
        /// </summary>
        private void cancelBtn_Click(object sender,
            System.EventArgs e)
        {
            this.Close();
        }
    }
}
```

6. FormMapInfoPix.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace MobileDeNS
{
    /// <summary>
    /// FormMapInfoPix is a helper form that allows the user to
    /// review currently loaded jpg pixel values for height and
    /// width in a read/write format.  It can be accessed through
    /// the Map menu found in MobileDeNS Form1 by selecting New Map,
    /// after entering data into the FormMapInfoLong page and
    /// selecting OK.  This is the Third form in the map parameter
    /// specification process.
    /// </summary>
    public class FormMapInfoPix : System.Windows.Forms.Form
    {
        private GPSHandler GPS;
        private System.Windows.Forms.Form refFormMapConfirmation;
        private System.Windows.Forms.Button cancelBtn;
        private System.Windows.Forms.Button okBtn;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.TextBox wPixCt;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox hPixCt;

        /// <summary>
        /// FormMapInfoPix constructor.  It requires an instance of
        /// GPSHandler to be passed as an argument.
        /// </summary>
        public FormMapInfoPix(GPSHandler inGPS)
        {
            //build the form
            InitializeComponent();

            //set pointer to GPSHandler argument
            GPS = inGPS;
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
```

```
        base.Dispose( disposing );
    }

    #region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        this.cancelBtn = new System.Windows.Forms.Button();
        this.okBtn = new System.Windows.Forms.Button();
        this.label5 = new System.Windows.Forms.Label();
        this.wPixCt = new System.Windows.Forms.TextBox();
        this.hPixCt = new System.Windows.Forms.TextBox();
        this.label1 = new System.Windows.Forms.Label();
        //
        // cancelBtn
        //
        this.cancelBtn.Location =
            new System.Drawing.Point(176, 128);
        this.cancelBtn.Size = new System.Drawing.Size(56, 24);
        this.cancelBtn.Text = "Cancel";
        this.cancelBtn.Click +=
            new System.EventHandler(this.cancelBtn_Click);
        //
        // okBtn
        //
        this.okBtn.Location = new System.Drawing.Point(136, 128);
        this.okBtn.Size = new System.Drawing.Size(32, 24);
        this.okBtn.Text = "OK";
        this.okBtn.Click +=
            new System.EventHandler(this.okBtn_Click);
        //
        // label5
        //
        this.label5.Location = new System.Drawing.Point(8, 16);
        this.label5.Size = new System.Drawing.Size(96, 16);
        this.label5.Text = "Height in Pixels";
        //
        // wPixCt
        //
        this.wPixCt.Location = new System.Drawing.Point(16, 80);
        this.wPixCt.Size = new System.Drawing.Size(48, 22);
        this.wPixCt.Text = "1195";
        //
        // hPixCt
        //
        this.hPixCt.Location = new System.Drawing.Point(16, 32);
        this.hPixCt.Size = new System.Drawing.Size(48, 22);
        this.hPixCt.Text = "910";
        //
        // label1
        //
        this.label1.Location = new System.Drawing.Point(8, 64);
        this.label1.Size = new System.Drawing.Size(96, 16);
        this.label1.Text = "Width in Pixels";
```

```
            //
            // FormMapInfoPix
            //
            this.Controls.Add(this.label1);
            this.Controls.Add(this.cancelBtn);
            this.Controls.Add(this.okBtn);
            this.Controls.Add(this.label5);
            this.Controls.Add(this.wPixCt);
            this.Controls.Add(this.hPixCt);
            this.Text = "Map Information - Pixel";

        }
        #endregion

        /// <summary>
        /// This button writes entered data into GPSHandler through
        /// the pointer.  It then opens the next form for data
        /// confirmation and closes itself.
        /// </summary>
        private void okBtn_Click(object sender, System.EventArgs e)
        {
            int pixInt;

            //convert string data for height to an integer
            pixInt = Convert.ToInt16(hPixCt.Text);

            //write integer data above to GPS
            GPS.mapper.pixelHeight = pixInt;

            //convert string data for width to an integer
            pixInt = Convert.ToInt16(wPixCt.Text);

            //write integer data above to GPS
            GPS.mapper.pixelWidth = pixInt;

            //open confirmation form
            this.refFormMapConfirmation =
                new FormMapConfirmation(GPS);
            this.refFormMapConfirmation.Show();

            this.Close();
        }

        /// <summary>
        /// Allows the user to cancel this process before values are
        /// written.
        /// </summary>
        private void cancelBtn_Click(object sender,
            System.EventArgs e)
        {
            this.Close();
        }
    }
}
```

7. GPSHandler.cs

```
using System;
using System.IO;
using System.Text;
using System.Windows.Forms;
using System.Drawing;
using System.Globalization;
using GPSAccess;

namespace MobileDeNS
{
    /// <summary>
    /// GPSHandler Class manages the GPS receiver connection and
    /// data retreival.  It also determines whether positions are
    /// within track tolerence and if they are valid positions,
    /// i.e., contained on the map.
    /// </summary>

    // Typical NMEA Sentence:
    //
    // $GPGGA,123519,4807.038,N,01131.000,E,
    //          1,08,0.9,545.4,M,46.9,M,,*47
    //  Where:
    //      GGA          Global Positioning System Fix Data
    //      123519       Fix taken at 12:35:19 UTC
    //      4807.038,N   Latitude 48 deg 07.038' N
    //      01131.000,E  Longitude 11 deg 31.000' E
    //      1            Fix quality: 0 = invalid
    //                   1 = GPS fix (SPS)
    //                   2 = DGPS fix
    //                   3 = PPS fix
    //                   4 = Real Time Kinematic
    //                   5 = Float RTK
    //                   6 = estimated (dead reckoning) (2.3 feature)
    //                   7 = Manual input mode
    //                   8 = Simulation mode
    //      08           Number of satellites being tracked
    //      0.9          Horizontal dilution of position
    //      545.4,M      Altitude, Meters, above mean sea level
    //      46.9,M       Height of geoid (mean sea level) above WGS84
    //                   ellipsoid
    //      (empty field) time in seconds since last DGPS update
    //      (empty field) DGPS station ID number
    //      *47          the checksum data, always begins with *


    public class GPSHandler
    {

        public MapHandler mapper = new MapHandler();
        public GPSpacket[] demoHistory = new GPSpacket[2000];
        public int demoIdx;
        public string sendDataStr;

        /// <summary>
        /// GPSHandler constructor.
```

```
/// </summary>
public GPSHandler()
{
    //set position array index to zero
    demoIdx = 0;
}

/// <summary>
/// Connect to GPS receiver
/// </summary>
public void start()
{
    GPS.Start();
}

/// <summary>
/// Disconnect from GPS receiver
/// </summary>
public void stop()
{
    GPS.Stop();
}

/// <summary>
/// Determine whether position is within track tolerence.
/// Returns boolean value of true or false to indicate
/// on-track staus
/// </summary>
public bool isOnTrack()
{
    GPSVector vectorA = new GPSVector();
    GPSVector vectorB = new GPSVector();
    GPSVector vectorC = new GPSVector();
    string strSideA;
    string strSideB;
    string strSideC;
    decimal sideA;
    decimal sideB;
    decimal sideC;
    decimal angCos;
    double angle;
    double distFromTrack;

    //determine vector of line from first waypoint to
    //  current position
    vectorA = GPS.GFC.GetVector(
        demoHistory[demoIdx - 1].getLatDecDeg(),
        demoHistory[demoIdx - 1].getLongDecDeg(),
        mapper.wpLatDecDeg[0],
        mapper.wpLongDecDeg[0]);
    //determine vector of line from current position to
    //  second waypoint
    vectorB = GPS.GFC.GetVector(
        demoHistory[demoIdx - 1].getLatDecDeg(),
        demoHistory[demoIdx - 1].getLongDecDeg(),
        mapper.wpLatDecDeg[1],
        mapper.wpLongDecDeg[1]);
```

```
        //determine vector of line from first waypoint to
        //  second waypoint
        vectorC = GPS.GFC.GetVector(mapper.wpLatDecDeg[0],
            mapper.wpLongDecDeg[0],
            mapper.wpLatDecDeg[1],
            mapper.wpLongDecDeg[1]);

        //get distances of all three vectors in yards
        strSideA = vectorA.Distance.Yards.ToString();
        strSideB = vectorB.Distance.Yards.ToString();
        strSideC = vectorC.Distance.Yards.ToString();

        //convert strings to decimal data types
        sideA = decimal.Parse(strSideA);
        sideB = decimal.Parse(strSideB);
        sideC = decimal.Parse(strSideC);

        //calculate the angle cosignof angle c-b
        angCos = (((sideA * sideA) - (sideB * sideB) -
            (sideC * sideC)) / ((-2) * (sideB) * (sideC)));

        //calculate the angle of c-b
        angle = Math.Acos((double)angCos);

        //calculate the distance from track by determining the
        //  height of the triangle
        distFromTrack = ((double)sideB) * Math.Sin(angle);

        //determine if position is within track tolerence; track
        //  tolerence is 50 yards.
        if (distFromTrack > 50)
        {
            return false;
        }
        else
        {
            return true;
        }
    }

    /// <summary>
    /// Retreives the time, latitude and longitude of last fix.
    /// </summary>
    public string getFixDetails()
    {
        string result = "";

        //no position exists, tell user and return null
        if (demoIdx == 0)
        {
            MessageBox.Show("No positional data exists at this" +
                " time");

            return result;
        }
        //position does exist, retreive last poition's data and
        //  return it as a string
```

```
    else
    {
        int curIdx = demoIdx - 1;

        result = demoHistory[curIdx].getTime() +
            demoHistory[curIdx].getLat() +
            demoHistory[curIdx].getLong();

        return result;
    }
}

/// <summary>
/// Polls the GPS receiver for new data and tests
/// whether or not it is contained on the current map.
/// </summary>
public bool getLiveData(/*Thread readThread*/)
{
    bool result;
    string type;
    string time;
    string str;
    string lat;
    string latID;
    string lon;
    string lonID;
    string satsUsed;
    int degIdx;
    int minIdx;
    int secIdx;
    string dataStr;
    int temp;

    //set number format
    NumberFormatInfo nfi =
        new CultureInfo("en-US", false).NumberFormat;
    nfi.NumberDecimalDigits = 3;

    //get GPS fix type
    type = "GGA";

    //get GPS time
    time = GPS.Current.UTCDateTime.ToLocalTime().
        ToShortTimeString().Substring(0,5);

    //get latitude
    str = GPS.Current.Latitude.Sexagesimal.ToString();

    //set latitude delimiters
    degIdx = str.IndexOf(" ");
    minIdx = str.IndexOf("'");
    secIdx = str.IndexOf("''");

    //pad latitude degrees appropriately
    if ((degIdx - 2) == 0)
    {
        lat = "0" + (GPS.Current.Latitude.DegreesPart);
```

```
}
else
{
    lat = (GPS.Current.Latitude.DegreesPart).ToString();
}

//pad latitude minutes appropriately
if ((minIdx - degIdx) == 2)
{
    lat = lat + "0" + (GPS.Current.Latitude.MinutesPart);
}
else
{
    lat = lat + (GPS.Current.Latitude.MinutesPart);

}

//get hemisphere
latID = str.Substring(str.Length - 1, 1);

//get longitude
str = GPS.Current.Longitude.Sexagesimal.ToString();

//set longitude delimiters
degIdx = str.IndexOf(" ");
minIdx = str.IndexOf("'");
secIdx = str.IndexOf("''");

//set temp value to longitude degrees
temp = GPS.Current.Longitude.DegreesPart;

//ignore neg values that may indicate hemisphere
if (temp < 0)
{
    lon = temp.ToString();
    lon = lon.Substring(1,lon.Length -1);
}
else
{
    lon = temp.ToString();
}

//pad longitude degrees appropriately
if ((degIdx - 2) == 0)
{
    lon = "00" + lon;
}
else if ((degIdx - 2) == 1)
{
    lon = "0" + lon;
}
else
{
    lon = lon;
}

//pad longitude minutes appropriately
```

```
if ((minIdx - degIdx) == 2)
{
    lon = lon + "0" + (GPS.Current.Longitude.MinutesPart);
}
else
{
    lon = lon + (GPS.Current.Longitude.MinutesPart);

}

//get longitude hemisphere
lonID = str.Substring(str.Length - 1, 1);

//get number of satellites used
satsUsed = GPS.Current.SatsUsed.ToString();

//pad satsUsed appropriately
if (int.Parse(satsUsed) < 10)
{
    satsUsed = "0" + satsUsed;
}

//get number portions of latitude and longitude strings
lat = lat.Substring(0,8);
lon = lon.Substring(0,9);

//build packet string
dataStr = type + lat + latID + lon + lonID + satsUsed;

//mark packet string as GPS packet
sendDataStr = "G" + dataStr;

//determine if position is on map
bool goodPosit = mapper.isValidPosit(float.Parse(lat),
                                     float.Parse(lon));

//is so
if (goodPosit)
{
    //make it a new packet with time
    GPSpacket pkt = new GPSpacket(time, dataStr);

    //set packet x and y pixel coordinates
    pkt.setPlot(mapper.calcX(pkt.getLongSecs(),
                             pkt.getLongMins()),
                mapper.calcY(pkt.getLatSecs(),
                             pkt.getLatMins()));

    //store the packet
    demoHistory[demoIdx] = pkt;

    //increment the packet storage index
    demoIdx++;

    result = true;
}
//if position not on map
```

```
    else
    {
        //tell the user
        MessageBox.Show("Last position read in not on NPS" +
                        "campus.");

        result = false;
    }

    return result;

}

/// <summary>
/// Determines if the GPS receiver connection is still active.
/// If so, it determines whether the position has changed
/// from the default position of 0.00 lat and 0.00 long,
/// which indicates awaiting satellite synchronization.
/// </summary>
public bool isRunning()
{
    if (GPS.Status.Equals(GPSStatus.Connected) &&
        ((GPS.Current.Latitude.Value != 0.00m) &&
        (GPS.Current.Longitude.Value != 0.00m)))
    {
        return true;
    }
    else
    {
        return false;
    }
}

/// <summary>
/// Parses NMEA strings into new GPSpackets.
/// </summary>
public void NMEAParser(string packet)
{
    // $GPGGA,123519,4807.038,N,01131.000,E,
    //          1,08,0.9,545.4,M,46.9,M,,*47

    string goodPacket;

    // Read 1st character from the packet
    goodPacket = packet.Substring(0,1);

    // if packet is good, parse for info, else print error
    //      message
    if (goodPacket == "$")
    {
        //is position on map
        bool goodPosit = mapper.isValidPosit(
                    float.Parse(packet.Substring(14,8)),
                    float.Parse(packet.Substring(25,9)));

        //if so
        if (goodPosit == true)
```

```
        {
            //create a new GPSpacket
            GPSpacket pkt = new GPSpacket(packet);

            //set packets x and y pixel coordinates
            pkt.setPlot(mapper.calcX(pkt.getLongSecs(),
                                     pkt.getLongMins()),
                        mapper.calcY(pkt.getLatSecs(),
                                     pkt.getLatMins()));

            //store packet and increment packet array index
            demoHistory[demoIdx] = pkt;
            demoIdx++;

        }
        //not on map
        else
        {
            //inform user
            MessageBox.Show("Last position read is not on" +
                            "NPS campus.");
        }
    }
    //Packet is bad
    else
    {
        //inform user
        MessageBox.Show("Bad packet received");
    }
}

/// <summary>
/// Returns the x pixel coordinate of a GPSpacket
/// </summary>
public int plotCurX()
{
    int x;
    int lastPositIdx;

    //set index to last position received
    lastPositIdx = demoIdx - 1;

    //get x coordinate
    x = demoHistory[lastPositIdx].getPlotX();

    return x;
}

/// <summary>
/// Returns the y pixel coordinate of a GPSpacket
/// </summary>
public int plotCurY()
{
    int y;
    int lastPositIdx;

    //set index to last position received
```

```
            lastPositIdx = demoIdx - 1;

            //get y coordinate
            y = demoHistory[lastPositIdx].getPlotY();

            return y;
        }

    }
}
```

8. GPSpacket.cs

```
using System;
using System.Windows.Forms;

namespace MobileDeNS
{
    /// <summary>
    /// GPSpacket Class serves as a storage and retreival vessel for
    /// GPS data to be used by the program.  An instance of this
    /// class represents a single GPS fix, thus mulitple fixes are
    /// managed elsewhere through the use of the appropriate data
    /// structures.
    /// </summary>
    public class GPSpacket
    {
        private string packetType;
        private string fixTime;
        private float latNum;
        private int latDeg;
        private int latMin;
        private float latSec;
        private string latHem;
        private float longNum;
        private int longDeg;
        private int longMin;
        private float longSec;
        private string longHem;
        private string satNum;
        private int plotX;
        private int plotY;
        private decimal latDecDeg;
        private decimal longDecDeg;

        /// <summary>
        /// Default GPSHandler constructor; creates null packet.
        /// </summary>
        public GPSpacket()
        {

        }

        /// <summary>
        /// GPSHandler constructor.  requires a string appropriately
        /// formatted for packet creation, including the time.
        /// </summary>
```

```
public GPSpacket(string packet)
{
    packetType = packet.Substring(3,3);
    fixTime = packet.Substring(7,2)+":"+
        packet.Substring(9,2)+":"+
        packet.Substring(11,2);
    latNum = float.Parse(packet.Substring(14,8));
    latDeg = int.Parse(packet.Substring(14,2));
    latMin = int.Parse(packet.Substring(16,2));
    latSec = float.Parse(packet.Substring(18,4)) * 60;
    latHem = packet.Substring(23,1);
    longNum = float.Parse(packet.Substring(25,9));
    longDeg = int.Parse(packet.Substring(25,3));
    longMin = int.Parse(packet.Substring(28,2));
    longSec = float.Parse(packet.Substring(30,4)) * 60;
    longHem = packet.Substring(35,1);
    satNum = packet.Substring(39,2);
    setDecDeg();
}

/// <summary>
/// GPSHandler constructor.  requires two strings
/// appropriately formatted for packet creation: the first
/// is the time in the format HH:MM:SS, while the second
/// is a packet excluding the time.
/// </summary>
public GPSpacket(string time, string data)
{
    fixTime = time;

    packetType = data.Substring(0,3);
    latNum = float.Parse(data.Substring(3,8));
    latDeg = int.Parse(data.Substring(3,2));
    latMin = int.Parse(data.Substring(5,2));
    latSec = float.Parse(data.Substring(7,4)) * 60;
    latHem = data.Substring(11,1);
    longNum = float.Parse(data.Substring(12,9));
    longDeg = int.Parse(data.Substring(12,3));
    longMin = int.Parse(data.Substring(15,2));
    longSec = float.Parse(data.Substring(17,4)) * 60;
    longHem = data.Substring(22,1);
    satNum = data.Substring(23,2);
    setDecDeg();
}

/// <summary>
/// Sets x and y pixel coordinates of fix.
/// </summary>
public void setPlot(int x, int y)
{
    plotX = x;
    plotY = y;
}

/// <summary>
/// returns x pixel coordinate.
/// </summary>
```

```
public int getPlotX()
{
    return plotX;
}

/// <summary>
/// Returns y pixel coordinate.
/// </summary>
public int getPlotY()
{
    return plotY;
}

/// <summary>
/// Returns latitude in string format seperated by
/// delimiters "degrees", "minutes", and "seconds" with one
/// letter hemisphere designator.
/// </summary>
public string getLat()
{
    string lat;

    lat = latDeg.ToString() + " degrees ";
    lat = lat + latMin.ToString() + " Minutes ";
    lat = lat + latSec.ToString() + " Seconds ";
    lat = lat + " " + latHem;

    return lat;
}

/// <summary>
/// Returns Latitude in decimal degrees in string format.
/// </summary>
public string getLatNum()
{
    return latNum.ToString();
}

/// <summary>
/// Returns latitude hemisphere.
/// </summary>
public string getLatHem()
{
    return latHem;
}

/// <summary>
/// Returns Longitude in decimal degrees in string format.
/// </summary>
public string getLongNum()
{
    return longNum.ToString();
}

/// <summary>
/// Returns longitude hemisphere.
/// </summary>
```

```
public string getLongHem()
{
    return longHem;
}

/// <summary>
/// Returns longitude in string format seperated by
/// delimiters "degrees", "minutes", and "seconds" with one
/// letter hemisphere designator.
/// </summary>
public string getLong()
{
    string lon;

    lon = longDeg.ToString() + " degrees ";
    lon = lon + longMin.ToString() + " Minutes ";
    lon = lon + longSec.ToString() + " Seconds ";
    lon = lon + longHem;

    return lon;
}

/// <summary>
/// Sets the packet type.
/// </summary>
public void setType(string type)
{
    packetType = type;
}

/// <summary>
/// Returns the packet type.
/// </summary>
public string getType()
{
    return packetType;
}

/// <summary>
/// Sets the packet fix time from string formatted
/// as HH:MM:SS.
/// </summary>
public void setTime(string time)
{
    fixTime = time.Substring(0,2) + ":" +
              time.Substring(2,2) + ":" +
              time.Substring(4,2);
}

/// <summary>
/// returns fix time as a string formatted as HHMMSS.
/// </summary>
public string getTime()
{
    return fixTime;
}
```

```
/// <summary>
/// Sets the latitude in decimal degrees and the
/// latitude hemisphere.
/// </summary>
public void setLat(float lat, string hem)
{
    latNum = lat;
    latHem = hem;
}

/// <summary>
/// Sets latitude and longitude decimal degree variables
/// </summary>
public void setDecDeg()
{
    decimal deg;
    decimal min;
    decimal sec;

    sec = (decimal)(getLatSecs() / 60);
    min = (decimal)getLatMins();
    min = (min + sec) / 60;
    deg = (decimal)(36 + min);
    latDecDeg = deg;

    sec = ((decimal)getLongSecs() / 60);
    min = (decimal)getLongMins();
    min = (min + sec) / 60;
    deg = (decimal)(121 + min);
    longDecDeg = deg;
}

/// <summary>
/// Returns latitude in decimal degrees.
/// </summary>
public decimal getLatDecDeg()
{
    return latDecDeg;
}

/// <summary>
/// Returns longitude in decimal degrees.
/// </summary>
public decimal getLongDecDeg()
{
    return longDecDeg;
}

/// <summary>
/// Returns latitude seconds in float format.
/// </summary>
public float getLatSecs()
{
    return latSec;
}

/// <summary>
```

```
        /// Returns latitude minutes in float format.
        /// </summary>
        public float getLatMins()
        {
            return latMin;
        }

        /// <summary>
        /// Sets the longitude in decimal degrees and the
        /// longitude hemisphere.
        /// </summary>
        public void setLong(float lon, string hem)
        {
            longNum = lon;
            longHem = hem;
        }

        /// <summary>
        /// Returns longitude seconds in float format.
        /// </summary>
        public float getLongSecs()
        {
            return longSec;
        }

        /// <summary>
        /// Returns longitude minutes in float format.
        /// </summary>
        public float getLongMins()
        {
            return longMin;
        }

        /// <summary>
        /// Sets the number of satellites used to obtain the fix.
        /// </summary>
        public void setSatNum(string sat)
        {
            satNum = sat;
        }

        /// <summary>
        /// Returns the number of satellites used to obtain the fix.
        /// </summary>
        public string getSatNum()
        {
            return satNum;
        }
    }
}
```

9. MapHandler.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;
```

```
namespace MobileDeNS
{
    /// <summary>
    /// MapHandler Class stores all currently loaded map parameters
    /// and implements hardcoded lat and long per pixel values.  It
    /// calculates the x and y pixel coordinates, and determines
    /// whether a new position is on the map.
    /// </summary>
    public class MapHandler
    {


        public float upperLat;
        public string upperNSHem;
        public float lowerLat;
        public string lowerNSHem;
        public float leftLong;
        public string leftEWHem;
        public float rightLong;
        public string rightEWHem;
        public float latPerPixel;
        public float longPerPixel;
        public int pixelHeight;
        public int pixelWidth;
        public string[] wayptLat = new string[20];
        public string[] wayptLong = new string[20];
        public decimal[] wpLatDecDeg = new decimal[20];
        public decimal[] wpLongDecDeg = new decimal[20];

        /// <summary>
        /// MapHandler constructor.
        /// </summary>
        public MapHandler()
        {
            upperLat = 3636.963F;
            lowerLat = 3634.526F;
            leftLong = 12154.167F;
            rightLong = 12150.966F;
            latPerPixel = .1606813F;
            longPerPixel = .1607197F;
            pixelHeight = 910;
            pixelWidth = 1195;
        }

        /// <summary>
        /// Receives longitudinal minutes and seconds in float
        /// format and converts them into the x pixel coordinate
        /// of the positional cursor which is returned as an integer.
        /// </summary>
        public int calcX(float lonSecs, float lonMins)
        {
            float longDif;
            float leftMinSec;
            float tempDif;
            int x;

            //convert the left longtitude minutes and seconds
```

```
    //  parameter to a combined decimal float
    leftMinSec = leftLong - 12100;
    tempDif = (leftMinSec % 1) * 60;
    tempDif = tempDif + ((leftMinSec - (tempDif / 60)) * 60);
    //subtract combined arguments for long  minutes and
    //  seconds from above result to get the distance from
    //  the left-most edge of the map
    longDif = tempDif - ((lonMins * 60) + lonSecs);
    //cast the above float to an integer to get the x pixel
    //  coordinate
    x = (int)(longDif / longPerPixel);

    //returns the x pixel coordinate
    return x;
}

/// <summary>
/// Receives latitudinal minutes and seconds in float
/// format and converts them into the y pixel coordinate
/// of the positional cursor which is returned as an integer.
/// </summary>
public int calcY(float latSecs, float latMins)
{
    float latDif;
    float upperMinSec;
    float tempDif;
    int y;

    //convert the upper latitude minutes and seconds
    //  parameter to a combined decimal float
    upperMinSec = upperLat - 3600;
    tempDif = (upperMinSec % 1) * 60;
    tempDif = tempDif + ((upperMinSec - (tempDif /60)) * 60);
    //subtract combined arguments for lat  minutes and
    //  seconds from above result to get the distance from
    //  the upper-most edge of the map
    latDif = tempDif - ((latMins * 60) + latSecs);
    //cast the above float to an integer to get the y pixel
    //  coordinate
    y = (int)(latDif / latPerPixel);

    //return the y pixel coordinate
    return y;
}

/// <summary>
/// determines if a given lat/long position is on the map
/// </summary>
/// <param name="lat"></param>
/// <param name="lon"></param>
/// <returns></returns>
public bool isValidPosit(float lat, float lon)
{
    //assume posit is not on the map
    bool result = false;

    //if lat and long are within the upper and lower,
```

```
            //  and left and right boundaries, repectively, posit
            //  is good
            if (lowerLat > lat || lat > upperLat)
                result = false;
            else if (rightLong < lon || lon < leftLong)
                result = true;

            //return result
            return result;
        }
    }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX II

A. OVERVIEW

Appendix II is intended to provide code listing for forms and classes used in the implementation of the DeNS server side of the DeNS prototype system. Each form or class will be detailed separately, with comments provided in-line. All code was programmed in the Microsoft C# language constrained by the Microsoft .NET Framework, and designed to run on a windows based PC platform utilizing the .NET Framework.

1. Form1.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace DeNS
{
    /// <summary>
    /// Main form for DeNS desktop application.  It instantiates the
    /// DeNSServer, MapHandler, and Plotter classes and contains all
    /// menus and the plotTimer that manages the drawing functions for
    /// DeNS.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.MainMenu mainMenu1;
        private System.Windows.Forms.PictureBox viewPort;
        private System.Windows.Forms.MenuItem fileMenu;
        private System.Windows.Forms.MenuItem mapMenu;
        private System.Windows.Forms.MenuItem connectMenu;
        private System.Windows.Forms.MenuItem fileMenuExit;
        private System.Windows.Forms.MenuItem mapMenuNew;
        private System.Windows.Forms.MenuItem mapMenuProp;
        private System.Windows.Forms.MenuItem mapMenuClr;
        private System.Windows.Forms.MenuItem connectMenuSrvr;
        private System.Windows.Forms.HScrollBar hBar;
        private System.Windows.Forms.VScrollBar vBar;
        private Form refNewMapProp;
        private Form refMapProp;
        private MapHandler mapper;
        private DeNSServer server;
        private Plotter plotter;
        private System.Windows.Forms.MenuItem menuItem1;
        private System.Windows.Forms.Button curPosition;
```

```
private System.Windows.Forms.MenuItem menuMapPlotter;
private System.Windows.Forms.Timer plotTimer;
private System.ComponentModel.IContainer components;

/// <summary>
/// Form1 constructor.
/// </summary>
public Form1()
{
    //Build the form
    InitializeComponent();

    //Instantiate helper classes
    mapper = new MapHandler();
    server = new DeNSServer(plotTimer);
    plotter = new Plotter(server, mapper);
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    this.mainMenu1 = new System.Windows.Forms.MainMenu();
    this.fileMenu = new System.Windows.Forms.MenuItem();
    this.fileMenuExit = new System.Windows.Forms.MenuItem();
    this.mapMenu = new System.Windows.Forms.MenuItem();
    this.mapMenuNew = new System.Windows.Forms.MenuItem();
    this.mapMenuProp = new System.Windows.Forms.MenuItem();
    this.mapMenuClr = new System.Windows.Forms.MenuItem();
    this.menuMapPlotter = new System.Windows.Forms.MenuItem();
    this.connectMenu = new System.Windows.Forms.MenuItem();
    this.connectMenuSrvr = new System.Windows.Forms.MenuItem();
    this.menuItem1 = new System.Windows.Forms.MenuItem();
    this.viewPort = new System.Windows.Forms.PictureBox();
    this.hBar = new System.Windows.Forms.HScrollBar();
    this.vBar = new System.Windows.Forms.VScrollBar();
    this.curPosition = new System.Windows.Forms.Button();
    this.plotTimer = new System.Windows.Forms.Timer
```

```
                                    (this.components);
this.SuspendLayout();
//
// mainMenu1
//
this.mainMenu1.MenuItems.AddRange
    (new System.Windows.Forms.MenuItem[]
                                        {this.fileMenu,
                                         this.mapMenu,
                                         this.connectMenu});
//
// fileMenu
//
this.fileMenu.Index = 0;
this.fileMenu.MenuItems.AddRange
    (new System.Windows.Forms.MenuItem[]
                                       {this.fileMenuExit});
this.fileMenu.Text = "File";
//
// fileMenuExit
//
this.fileMenuExit.Index = 0;
this.fileMenuExit.Text = "Exit";
this.fileMenuExit.Click += new System.EventHandler
                                (this.fileMenuExit_Click);
//
// mapMenu
//
this.mapMenu.Index = 1;
this.mapMenu.MenuItems.AddRange
    (new System.Windows.Forms.MenuItem[] {this.mapMenuNew,
                                        this.mapMenuProp,
                                        this.mapMenuClr,
                                        this.menuMapPlotter});
this.mapMenu.Text = "Map";
//
// mapMenuNew
//
this.mapMenuNew.Index = 0;
this.mapMenuNew.Text = "New Map";
this.mapMenuNew.Click += new System.EventHandler
                                    (this.mapMenuNew_Click);
//
// mapMenuProp
//
this.mapMenuProp.Index = 1;
this.mapMenuProp.Text = "Properties";
this.mapMenuProp.Click += new System.EventHandler
                                (this.mapMenuProp_Click);
//
// mapMenuClr
//
this.mapMenuClr.Index = 2;
this.mapMenuClr.Text = "Clear Map";
this.mapMenuClr.Click += new System.EventHandler
                                    (this.mapMenuClr_Click);
//
```

```
// menuMapPlotter
//
this.menuMapPlotter.Index = 3;
this.menuMapPlotter.Text = "Start Plotter";
this.menuMapPlotter.Click += new System.EventHandler
                             (this.menuMapPlotter_Click);
//
// connectMenu
//
this.connectMenu.Index = 2;
this.connectMenu.MenuItems.AddRange
    (new System.Windows.Forms.MenuItem[] {
                                    this.connectMenuSrvr,
                                    this.menuItem1});
this.connectMenu.Text = "Connect";
//
// connectMenuSrvr
//
this.connectMenuSrvr.Enabled = false;
this.connectMenuSrvr.Index = 0;
this.connectMenuSrvr.Text = "Start Server";
this.connectMenuSrvr.Click += new System.EventHandler
                            (this.connectMenuSrvr_Click);
//
// menuItem1
//
this.menuItem1.Index = 1;
this.menuItem1.Text = "Test Server";
this.menuItem1.Click += new System.EventHandler
                                 (this.menuItem1_Click);
//
// viewPort
//
this.viewPort.Location = new System.Drawing.Point(0, 0);
this.viewPort.Name = "viewPort";
this.viewPort.SizeMode =
    System.Windows.Forms.PictureBoxSizeMode.AutoSize;
this.viewPort.TabIndex = 0;
this.viewPort.TabStop = false;
//
// hBar
//
this.hBar.Dock = System.Windows.Forms.DockStyle.Bottom;
this.hBar.Location = new System.Drawing.Point(0, 561);
this.hBar.Name = "hBar";
this.hBar.Size = new System.Drawing.Size(896, 16);
this.hBar.TabIndex = 1;
this.hBar.ValueChanged += new System.EventHandler
                                (this.hBar_ValueChanged);
//
// vBar
//
this.vBar.Dock = System.Windows.Forms.DockStyle.Right;
this.vBar.Location = new System.Drawing.Point(880, 0);
this.vBar.Name = "vBar";
this.vBar.Size = new System.Drawing.Size(16, 561);
this.vBar.TabIndex = 2;
```

```
    this.vBar.ValueChanged += new System.EventHandler
                                    (this.vBar_ValueChanged);
    //
    // curPosition
    //
    this.curPosition.BackColor = System.Drawing.Color.Red;
    this.curPosition.Location =
                        new System.Drawing.Point(32, 16);
    this.curPosition.Name = "curPosition";
    this.curPosition.Size = new System.Drawing.Size(8, 8);
    this.curPosition.TabIndex = 3;
    this.curPosition.Visible = false;
    //
    // plotTimer
    //
    this.plotTimer.Interval = 5000;
    this.plotTimer.Tick += new System.EventHandler
                                    (this.plotTimer_Tick);
    //
    // Form1
    //
    this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
    this.ClientSize = new System.Drawing.Size(896, 577);
    this.Controls.Add(this.curPosition);
    this.Controls.Add(this.vBar);
    this.Controls.Add(this.hBar);
    this.Controls.Add(this.viewPort);
    this.FormBorderStyle =
        System.Windows.Forms.FormBorderStyle.Fixed3D;
    this.MaximizeBox = false;
    this.Menu = this.mainMenu1;
    this.Name = "Form1";
    this.StartPosition =
        System.Windows.Forms.FormStartPosition.CenterScreen;
    this.Text = "DeNS";
    this.ResumeLayout(false);

}
#endregion

/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
    Application.Run(new Form1());
}

/// <summary>
/// Redraws viewPort if horizontally scrolled
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void hBar_ValueChanged(object sender,
    System.EventArgs e)
{
```

```
            //move left edge
            viewPort.Left = -hBar.Value;

            //show scrolling
            viewPort.Refresh();
    }

    /// <summary>
    /// Redraws viewPort if vertically scrolled
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void vBar_ValueChanged(object sender,
        System.EventArgs e)
    {
            //move top edge
            viewPort.Top = -vBar.Value;

            //show scrolling
            viewPort.Refresh();
    }
    /// <summary>
    /// Closes the application and frees all resources.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void fileMenuExit_Click(object sender,
        System.EventArgs e)
    {
        Application.Exit();
    }
    /// <summary>
    /// Allows user to load a new map and it's parameters.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void mapMenuNew_Click(object sender,
        System.EventArgs e)
    {
        Bitmap newMap;

        //Calls getMap method from MapHandler class to retreive
        //  map name and path
        newMap = new Bitmap(mapper.getMap());

        //load the map file into viewPort
        this.viewPort.Image = newMap;
        // + 16 allows all to be viewed considering hBar
        this.viewPort.Height = newMap.Height + 16;
        // + 16 allows all to be viewed considering vBar
        this.viewPort.Width = newMap.Width + 16;
        //visible viewport height + 16 = 577
        this.vBar.Maximum = newMap.Height - 561;
        //visible viewport width + 16 = 880
        this.hBar.Maximum = newMap.Width - 874;

        //allows the user to enter the map parameters for the new
```

```
        //  map.  Default are those used for the Monterey Bay
        //  test map.
        this.refNewMapProp = new FormNewMapProp(mapper);
        this.refNewMapProp.Show();
    }
    /// <summary>
    /// Allows the user to review the map parametrs being used
    /// including pixels per lat and long.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void mapMenuProp_Click(object sender,
        System.EventArgs e)
    {
        this.refMapProp = new FormMapProp(mapper);
        this.refMapProp.Show();
    }
    /// <summary>
    /// Used in early stages of development to call the
    /// testPosDraw method of DeNSServer which loads dummy posits
    /// for testing and verification purposes.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void connectMenuSrvr_Click(object sender,
        System.EventArgs e)
    {
        server.testPosDraw();
    }
    /// <summary>
    /// Used in early stages of development to test dummy posits
    /// previously loaded for testing and verification purposes.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void menuItem1_Click(object sender,
        System.EventArgs e)
    {
        Point scrPos;
        string msg;

        //there is a new position
        if(server.newPosExists())
        {
            //get the posit, calculate the screen position in
            //pixels, and draw it
            scrPos = plotter.calcScrPos();
            this.curPosition.Location = scrPos;
            this.curPosition.Visible = true;
        }
        //there is a new message
        else if(server.newMsgExists())
        {
            //get the new message and display it
            msg = server.getNewMsg();
            MessageBox.Show(msg);
        }
```

```
    //no new positions nor messages exist
    else
    {
        MessageBox.Show("No new positions nor messages" +
                        " exist at this time.");
    }
}
/// <summary>
/// Clears the viewPort of any previously drawn positions
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void mapMenuClr_Click(object sender,
    System.EventArgs e)
{
    this.curPosition.Visible = false;
}
/// <summary>
/// Starts or stops the plotter depending on its current
/// operational status, as determined by the menu item's
/// text description.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void menuMapPlotter_Click(object sender,
    System.EventArgs e)
{

    if(menuMapPlotter.Text == "Start Plotter")
    {
        //start plotter: change the menu text, and enable
        //  plot timer
        menuMapPlotter.Text = "Stop Plotter";
        plotTimer.Enabled = true;
    }
    else
    {
        //stop plotter: change the menu text, and disable
        //  plot timer
        menuMapPlotter.Text = "Start Plotter";
        plotTimer.Enabled = false;
    }
}
/// <summary>
/// executes a complete event cycle of the plotTimer every
/// five seconds.  Checks for new positions and associated
/// messages, then just messages, and if none of either exist,
/// it informs the user.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void plotTimer_Tick(object sender,
    System.EventArgs e)
{
    Point scrPos;
    string msg;
```

```
            //does new position exist?
            if(server.newPosExists())
            {
                //calculate screen position and set the cursor
                //  location
                scrPos = plotter.calcScrPos();
                this.curPosition.Location = scrPos;
                //determine new positions relation to track
                if(server.positGreen == true)
                {
                    //tolerence has not been exceeded
                    this.curPosition.BackColor = Color.Green;
                }
                else
                {
                    //tolerence has been exceeded
                    this.curPosition.BackColor = Color.Red;
                }
                //draw the new position
                this.curPosition.Visible = true;

                //if position exceeded tolerence a new message will
                //  exist
                if(server.newMsgExists())
                {
                    //display warning after position is drawn
                    msg = server.getNewMsg();
                    MessageBox.Show(msg);
                }
            }
            //does new message exist?
            else if(server.newMsgExists())
            {
                //get and display new message
                msg = server.getNewMsg();
                MessageBox.Show(msg);
            }
            //no new positions nor messages exist
            else
            {
                MessageBox.Show("No new positions nor messages" +
                                " exist at this time.");
            }
        }
    }
}
```

2. FormMapProp.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace DeNS
{
```

```
/// <summary>
/// This helper form displays currently loaded map parameters
/// for review to include pixels per lat and longitude.  It is
/// read only, allowing no changes to be made.
/// </summary>
public class FormMapProp : System.Windows.Forms.Form
{
    private System.Windows.Forms.Label lblInst;
    private System.Windows.Forms.Label lblUpperLat;
    private System.Windows.Forms.Label label2;
    private System.Windows.Forms.Label label3;
    private System.Windows.Forms.Label lblUpperDeg;
    private System.Windows.Forms.TextBox tbUpperDeg;
    private System.Windows.Forms.TextBox tbUpperMin;
    private System.Windows.Forms.TextBox tbUpperSec;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.Label label4;
    private System.Windows.Forms.Label label5;
    private System.Windows.Forms.Label label6;
    private System.Windows.Forms.Label label7;
    private System.Windows.Forms.Label label8;
    private System.Windows.Forms.Label label9;
    private System.Windows.Forms.Label label10;
    private System.Windows.Forms.Label label11;
    private System.Windows.Forms.Label label12;
    private System.Windows.Forms.Label label13;
    private System.Windows.Forms.Label label14;
    private System.Windows.Forms.Button button1;
    private System.Windows.Forms.TextBox tbLowerSec;
    private System.Windows.Forms.TextBox tbLowerMin;
    private System.Windows.Forms.TextBox tbLowerDeg;
    private System.Windows.Forms.TextBox tbLeftDeg;
    private System.Windows.Forms.TextBox tbLeftSec;
    private System.Windows.Forms.TextBox tbLeftMin;
    private System.Windows.Forms.TextBox tbRightSec;
    private System.Windows.Forms.TextBox tbRightMin;
    private System.Windows.Forms.TextBox tbRightDeg;
    private MapHandler myMap;
    private System.Windows.Forms.Label label15;
    private System.Windows.Forms.Label label16;
    /// <summary>
    /// Required designer variable.
    /// </summary>
    private System.ComponentModel.Container components = null;
    /// <summary>
    /// FormMapProp constructor.  Receives an instance of the
    /// current MapHandler as an argument from which to draw
    /// current parameters.
    /// </summary>
    public FormMapProp(MapHandler mapper)
    {
        InitializeComponent();

        //sets pointer to MapHandler argument
        myMap = mapper;

        //loads parameters
```

```
    this.tbUpperDeg.Text = myMap.upperDeg;
    this.tbUpperMin.Text = myMap.upperMin;
    this.tbUpperSec.Text = myMap.upperSec;
    this.tbLowerSec.Text = myMap.lowerSec;
    this.tbLowerMin.Text = myMap.lowerMin;
    this.tbLowerDeg.Text = myMap.lowerDeg;
    this.tbLeftSec.Text = myMap.leftSec;
    this.tbLeftMin.Text = myMap.leftMin;
    this.tbLeftDeg.Text = myMap.leftDeg;
    this.tbRightSec.Text = myMap.rightSec;
    this.tbRightMin.Text = myMap.rightMin;
    this.tbRightDeg.Text = myMap.rightDeg;
    this.label15.Text = "Longitude per pixel: " +
                    myMap.longPerPixel.ToString();
    this.label16.Text = "Latitude per pixel: " +
                    myMap.latPerPixel.ToString();
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if(components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.lblInst = new System.Windows.Forms.Label();
    this.lblUpperLat = new System.Windows.Forms.Label();
    this.lblUpperDeg = new System.Windows.Forms.Label();
    this.label2 = new System.Windows.Forms.Label();
    this.label3 = new System.Windows.Forms.Label();
    this.tbUpperDeg = new System.Windows.Forms.TextBox();
    this.tbUpperMin = new System.Windows.Forms.TextBox();
    this.tbUpperSec = new System.Windows.Forms.TextBox();
    this.tbLowerSec = new System.Windows.Forms.TextBox();
    this.tbLowerMin = new System.Windows.Forms.TextBox();
    this.tbLowerDeg = new System.Windows.Forms.TextBox();
    this.label1 = new System.Windows.Forms.Label();
    this.label4 = new System.Windows.Forms.Label();
    this.label5 = new System.Windows.Forms.Label();
    this.label6 = new System.Windows.Forms.Label();
    this.tbLeftSec = new System.Windows.Forms.TextBox();
    this.tbLeftMin = new System.Windows.Forms.TextBox();
```

```
this.tbLeftDeg = new System.Windows.Forms.TextBox();
this.label7 = new System.Windows.Forms.Label();
this.label8 = new System.Windows.Forms.Label();
this.label9 = new System.Windows.Forms.Label();
this.label10 = new System.Windows.Forms.Label();
this.tbRightSec = new System.Windows.Forms.TextBox();
this.tbRightMin = new System.Windows.Forms.TextBox();
this.tbRightDeg = new System.Windows.Forms.TextBox();
this.label11 = new System.Windows.Forms.Label();
this.label12 = new System.Windows.Forms.Label();
this.label13 = new System.Windows.Forms.Label();
this.label14 = new System.Windows.Forms.Label();
this.button1 = new System.Windows.Forms.Button();
this.label15 = new System.Windows.Forms.Label();
this.label16 = new System.Windows.Forms.Label();
this.SuspendLayout();
//
// lblInst
//
this.lblInst.Location = new System.Drawing.Point(8, 16);
this.lblInst.Name = "lblInst";
this.lblInst.Size = new System.Drawing.Size(392, 16);
this.lblInst.TabIndex = 0;
this.lblInst.Text = "Please enter the following " +
    "information regarding the map your have chosen:";
//
// lblUpperLat
//
this.lblUpperLat.Location =
    new System.Drawing.Point(24, 48);
this.lblUpperLat.Name = "lblUpperLat";
this.lblUpperLat.Size =
    new System.Drawing.Size(88, 16);
this.lblUpperLat.TabIndex = 0;
this.lblUpperLat.Text = "Upper Latitude:";
//
// lblUpperDeg
//
this.lblUpperDeg.Location =
    new System.Drawing.Point(112, 80);
this.lblUpperDeg.Name = "lblUpperDeg";
this.lblUpperDeg.Size = new System.Drawing.Size(48, 16);
this.lblUpperDeg.TabIndex = 0;
this.lblUpperDeg.Text = "Degrees";
//
// label2
//
this.label2.Location = new System.Drawing.Point(232, 80);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(48, 16);
this.label2.TabIndex = 0;
this.label2.Text = "Minutes";
//
// label3
//
this.label3.Location = new System.Drawing.Point(352, 80);
this.label3.Name = "label3";
```

```
this.label3.Size = new System.Drawing.Size(56, 16);
this.label3.TabIndex = 0;
this.label3.Text = "Seconds";
//
// tbUpperDeg
//
this.tbUpperDeg.Location =
    new System.Drawing.Point(56, 72);
this.tbUpperDeg.Name = "tbUpperDeg";
this.tbUpperDeg.ReadOnly = true;
this.tbUpperDeg.Size = new System.Drawing.Size(48, 20);
this.tbUpperDeg.TabIndex = 1;
this.tbUpperDeg.Text = "00";
this.tbUpperDeg.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbUpperMin
//
this.tbUpperMin.Location =
    new System.Drawing.Point(176, 72);
this.tbUpperMin.Name = "tbUpperMin";
this.tbUpperMin.ReadOnly = true;
this.tbUpperMin.Size = new System.Drawing.Size(48, 20);
this.tbUpperMin.TabIndex = 2;
this.tbUpperMin.Text = "00";
this.tbUpperMin.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbUpperSec
//
this.tbUpperSec.Location =
    new System.Drawing.Point(296, 72);
this.tbUpperSec.Name = "tbUpperSec";
this.tbUpperSec.ReadOnly = true;
this.tbUpperSec.Size = new System.Drawing.Size(48, 20);
this.tbUpperSec.TabIndex = 3;
this.tbUpperSec.Text = "00.0";
this.tbUpperSec.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbLowerSec
//
this.tbLowerSec.Location =
    new System.Drawing.Point(296, 152);
this.tbLowerSec.Name = "tbLowerSec";
this.tbLowerSec.ReadOnly = true;
this.tbLowerSec.Size = new System.Drawing.Size(48, 20);
this.tbLowerSec.TabIndex = 6;
this.tbLowerSec.Text = "00.0";
this.tbLowerSec.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbLowerMin
//
this.tbLowerMin.Location =
    new System.Drawing.Point(176, 152);
this.tbLowerMin.Name = "tbLowerMin";
```

```
this.tbLowerMin.ReadOnly = true;
this.tbLowerMin.Size = new System.Drawing.Size(48, 20);
this.tbLowerMin.TabIndex = 5;
this.tbLowerMin.Text = "00";
this.tbLowerMin.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbLowerDeg
//
this.tbLowerDeg.Location =
    new System.Drawing.Point(56, 152);
this.tbLowerDeg.Name = "tbLowerDeg";
this.tbLowerDeg.ReadOnly = true;
this.tbLowerDeg.Size = new System.Drawing.Size(48, 20);
this.tbLowerDeg.TabIndex = 4;
this.tbLowerDeg.Text = "00";
this.tbLowerDeg.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// label1
//
this.label1.Location = new System.Drawing.Point(352, 160);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(56, 16);
this.label1.TabIndex = 0;
this.label1.Text = "Seconds";
//
// label4
//
this.label4.Location = new System.Drawing.Point(232, 160);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(48, 16);
this.label4.TabIndex = 0;
this.label4.Text = "Minutes";
//
// label5
//
this.label5.Location = new System.Drawing.Point(112, 160);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(48, 16);
this.label5.TabIndex = 0;
this.label5.Text = "Degrees";
//
// label6
//
this.label6.Location = new System.Drawing.Point(24, 128);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(88, 16);
this.label6.TabIndex = 0;
this.label6.Text = "Lower Latitude:";
//
// tbLeftSec
//
this.tbLeftSec.Location =
    new System.Drawing.Point(296, 232);
this.tbLeftSec.Name = "tbLeftSec";
this.tbLeftSec.ReadOnly = true;
```

```
this.tbLeftSec.Size = new System.Drawing.Size(48, 20);
this.tbLeftSec.TabIndex = 9;
this.tbLeftSec.Text = "00.0";
this.tbLeftSec.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbLeftMin
//
this.tbLeftMin.Location =
    new System.Drawing.Point(176, 232);
this.tbLeftMin.Name = "tbLeftMin";
this.tbLeftMin.ReadOnly = true;
this.tbLeftMin.Size = new System.Drawing.Size(48, 20);
this.tbLeftMin.TabIndex = 8;
this.tbLeftMin.Text = "00";
this.tbLeftMin.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbLeftDeg
//
this.tbLeftDeg.Location =
    new System.Drawing.Point(56, 232);
this.tbLeftDeg.Name = "tbLeftDeg";
this.tbLeftDeg.ReadOnly = true;
this.tbLeftDeg.Size = new System.Drawing.Size(48, 20);
this.tbLeftDeg.TabIndex = 7;
this.tbLeftDeg.Text = "000";
this.tbLeftDeg.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// label7
//
this.label7.Location = new System.Drawing.Point(352, 240);
this.label7.Name = "label7";
this.label7.Size = new System.Drawing.Size(56, 16);
this.label7.TabIndex = 0;
this.label7.Text = "Seconds";
//
// label8
//
this.label8.Location = new System.Drawing.Point(232, 240);
this.label8.Name = "label8";
this.label8.Size = new System.Drawing.Size(48, 16);
this.label8.TabIndex = 0;
this.label8.Text = "Minutes";
//
// label9
//
this.label9.Location = new System.Drawing.Point(112, 240);
this.label9.Name = "label9";
this.label9.Size = new System.Drawing.Size(48, 16);
this.label9.TabIndex = 0;
this.label9.Text = "Degrees";
//
// label10
//
this.label10.Location = new System.Drawing.Point(24, 208);
```

```
this.label10.Name = "label10";
this.label10.Size = new System.Drawing.Size(80, 16);
this.label10.TabIndex = 0;
this.label10.Text = "Left Longitude:";
//
// tbRightSec
//
this.tbRightSec.Location =
    new System.Drawing.Point(296, 312);
this.tbRightSec.Name = "tbRightSec";
this.tbRightSec.ReadOnly = true;
this.tbRightSec.Size = new System.Drawing.Size(48, 20);
this.tbRightSec.TabIndex = 12;
this.tbRightSec.Text = "00.0";
this.tbRightSec.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbRightMin
//
this.tbRightMin.Location =
    new System.Drawing.Point(176, 312);
this.tbRightMin.Name = "tbRightMin";
this.tbRightMin.ReadOnly = true;
this.tbRightMin.Size = new System.Drawing.Size(48, 20);
this.tbRightMin.TabIndex = 11;
this.tbRightMin.Text = "00";
this.tbRightMin.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbRightDeg
//
this.tbRightDeg.Location =
    new System.Drawing.Point(56, 312);
this.tbRightDeg.Name = "tbRightDeg";
this.tbRightDeg.ReadOnly = true;
this.tbRightDeg.Size = new System.Drawing.Size(48, 20);
this.tbRightDeg.TabIndex = 10;
this.tbRightDeg.Text = "000";
this.tbRightDeg.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// label11
//
this.label11.Location =
    new System.Drawing.Point(352, 320);
this.label11.Name = "label11";
this.label11.Size = new System.Drawing.Size(56, 16);
this.label11.TabIndex = 0;
this.label11.Text = "Seconds";
//
// label12
//
this.label12.Location =
    new System.Drawing.Point(232, 320);
this.label12.Name = "label12";
this.label12.Size = new System.Drawing.Size(48, 16);
this.label12.TabIndex = 0;
```

```
this.label12.Text = "Minutes";
//
// label13
//
this.label13.Location =
    new System.Drawing.Point(112, 320);
this.label13.Name = "label13";
this.label13.Size = new System.Drawing.Size(48, 16);
this.label13.TabIndex = 0;
this.label13.Text = "Degrees";
//
// label14
//
this.label14.Location =
    new System.Drawing.Point(24, 288);
this.label14.Name = "label14";
this.label14.Size = new System.Drawing.Size(88, 16);
this.label14.TabIndex = 0;
this.label14.Text = "Right Longitude:";
//
// button1
//
this.button1.Location =
    new System.Drawing.Point(328, 424);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(72, 24);
this.button1.TabIndex = 13;
this.button1.Text = "OK";
this.button1.Click +=
    new System.EventHandler(this.button1_Click);
//
// label15
//
this.label15.Location = new System.Drawing.Point(24, 392);
this.label15.Name = "label15";
this.label15.Size = new System.Drawing.Size(264, 16);
this.label15.TabIndex = 14;
this.label15.Text = "Longitude per pixel: ";
//
// label16
//
this.label16.Location = new System.Drawing.Point(24, 368);
this.label16.Name = "label16";
this.label16.Size = new System.Drawing.Size(264, 16);
this.label16.TabIndex = 15;
this.label16.Text = "Latitude per pixel: ";
//
// FormMapProp
//
this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.ClientSize = new System.Drawing.Size(416, 462);
this.Controls.Add(this.label16);
this.Controls.Add(this.label15);
this.Controls.Add(this.button1);
this.Controls.Add(this.tbRightSec);
this.Controls.Add(this.tbRightMin);
this.Controls.Add(this.tbRightDeg);
```

```
            this.Controls.Add(this.label11);
            this.Controls.Add(this.label12);
            this.Controls.Add(this.label13);
            this.Controls.Add(this.label14);
            this.Controls.Add(this.tbLeftSec);
            this.Controls.Add(this.tbLeftMin);
            this.Controls.Add(this.tbLeftDeg);
            this.Controls.Add(this.label7);
            this.Controls.Add(this.label8);
            this.Controls.Add(this.label9);
            this.Controls.Add(this.label10);
            this.Controls.Add(this.tbLowerSec);
            this.Controls.Add(this.tbLowerMin);
            this.Controls.Add(this.tbLowerDeg);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.label4);
            this.Controls.Add(this.label5);
            this.Controls.Add(this.label6);
            this.Controls.Add(this.tbUpperSec);
            this.Controls.Add(this.tbUpperMin);
            this.Controls.Add(this.tbUpperDeg);
            this.Controls.Add(this.label3);
            this.Controls.Add(this.label2);
            this.Controls.Add(this.lblUpperDeg);
            this.Controls.Add(this.lblUpperLat);
            this.Controls.Add(this.lblInst);
            this.Name = "FormMapProp";
            this.StartPosition =
                System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "New Map Properties";
            this.ResumeLayout(false);

        }
        #endregion
        /// <summary>
        /// button1 action event handler.  Simply closes form.
        /// </summary>
        private void button1_Click(object sender, System.EventArgs e)
        {
            this.Close();
        }
    }
}
```

3. FormNewMapProp.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace DeNS
{
    /// <summary>
    /// FormNewMapProp is a helper form that allows the user to enter
    /// map parameters for a new map to be loaded.
```

```
/// </summary>
public class FormNewMapProp : System.Windows.Forms.Form
{
    private System.Windows.Forms.Label lblInst;
    private System.Windows.Forms.Label lblUpperLat;
    private System.Windows.Forms.Label label2;
    private System.Windows.Forms.Label label3;
    private System.Windows.Forms.Label lblUpperDeg;
    private System.Windows.Forms.TextBox tbUpperDeg;
    private System.Windows.Forms.TextBox tbUpperMin;
    private System.Windows.Forms.TextBox tbUpperSec;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.Label label4;
    private System.Windows.Forms.Label label5;
    private System.Windows.Forms.Label label6;
    private System.Windows.Forms.Label label7;
    private System.Windows.Forms.Label label8;
    private System.Windows.Forms.Label label9;
    private System.Windows.Forms.Label label10;
    private System.Windows.Forms.Label label11;
    private System.Windows.Forms.Label label12;
    private System.Windows.Forms.Label label13;
    private System.Windows.Forms.Label label14;
    private System.Windows.Forms.Button button1;
    private System.Windows.Forms.Button button2;
    private System.Windows.Forms.TextBox tbLowerSec;
    private System.Windows.Forms.TextBox tbLowerMin;
    private System.Windows.Forms.TextBox tbLowerDeg;
    private System.Windows.Forms.TextBox tbLeftDeg;
    private System.Windows.Forms.TextBox tbLeftSec;
    private System.Windows.Forms.TextBox tbLeftMin;
    private System.Windows.Forms.TextBox tbRightSec;
    private System.Windows.Forms.TextBox tbRightMin;
    private System.Windows.Forms.TextBox tbRightDeg;
    private MapHandler myMap;
    /// <summary>
    /// Required designer variable.
    /// </summary>
    private System.ComponentModel.Container components = null;

    /// <summary>
    /// FormNewMapProp Constructor.  receives an instance of
    /// the current MapHandler being used by the application
    /// as an argument from which to read and write map
    /// parameters from and to, respectively.
    /// </summary>
    public FormNewMapProp(MapHandler mapper)
    {
        InitializeComponent();

        //set pointer to the current instance of MapHandler
        myMap = mapper;
    }

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
```

```
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if(components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.lblInst = new System.Windows.Forms.Label();
    this.lblUpperLat = new System.Windows.Forms.Label();
    this.lblUpperDeg = new System.Windows.Forms.Label();
    this.label2 = new System.Windows.Forms.Label();
    this.label3 = new System.Windows.Forms.Label();
    this.tbUpperDeg = new System.Windows.Forms.TextBox();
    this.tbUpperMin = new System.Windows.Forms.TextBox();
    this.tbUpperSec = new System.Windows.Forms.TextBox();
    this.tbLowerSec = new System.Windows.Forms.TextBox();
    this.tbLowerMin = new System.Windows.Forms.TextBox();
    this.tbLowerDeg = new System.Windows.Forms.TextBox();
    this.label1 = new System.Windows.Forms.Label();
    this.label4 = new System.Windows.Forms.Label();
    this.label5 = new System.Windows.Forms.Label();
    this.label6 = new System.Windows.Forms.Label();
    this.tbLeftSec = new System.Windows.Forms.TextBox();
    this.tbLeftMin = new System.Windows.Forms.TextBox();
    this.tbLeftDeg = new System.Windows.Forms.TextBox();
    this.label7 = new System.Windows.Forms.Label();
    this.label8 = new System.Windows.Forms.Label();
    this.label9 = new System.Windows.Forms.Label();
    this.label10 = new System.Windows.Forms.Label();
    this.tbRightSec = new System.Windows.Forms.TextBox();
    this.tbRightMin = new System.Windows.Forms.TextBox();
    this.tbRightDeg = new System.Windows.Forms.TextBox();
    this.label11 = new System.Windows.Forms.Label();
    this.label12 = new System.Windows.Forms.Label();
    this.label13 = new System.Windows.Forms.Label();
    this.label14 = new System.Windows.Forms.Label();
    this.button1 = new System.Windows.Forms.Button();
    this.button2 = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // lblInst
    //
    this.lblInst.Location = new System.Drawing.Point(8, 16);
    this.lblInst.Name = "lblInst";
    this.lblInst.Size = new System.Drawing.Size(392, 16);
```

```
this.lblInst.TabIndex = 0;
this.lblInst.Text = "Please enter the following " +
    "information regarding the map your have chosen:";
//
// lblUpperLat
//
this.lblUpperLat.Location =
    new System.Drawing.Point(24, 48);
this.lblUpperLat.Name = "lblUpperLat";
this.lblUpperLat.Size = new System.Drawing.Size(88, 16);
this.lblUpperLat.TabIndex = 0;
this.lblUpperLat.Text = "Upper Latitude:";
//
// lblUpperDeg
//
this.lblUpperDeg.Location =
    new System.Drawing.Point(112, 80);
this.lblUpperDeg.Name = "lblUpperDeg";
this.lblUpperDeg.Size = new System.Drawing.Size(48, 16);
this.lblUpperDeg.TabIndex = 0;
this.lblUpperDeg.Text = "Degrees";
//
// label2
//
this.label2.Location = new System.Drawing.Point(232, 80);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(48, 16);
this.label2.TabIndex = 0;
this.label2.Text = "Minutes";
//
// label3
//
this.label3.Location = new System.Drawing.Point(352, 80);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(56, 16);
this.label3.TabIndex = 0;
this.label3.Text = "Seconds";
//
// tbUpperDeg
//
this.tbUpperDeg.Location =
    new System.Drawing.Point(56, 72);
this.tbUpperDeg.Name = "tbUpperDeg";
this.tbUpperDeg.Size = new System.Drawing.Size(48, 20);
this.tbUpperDeg.TabIndex = 1;
this.tbUpperDeg.Text = "36";
this.tbUpperDeg.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbUpperMin
//
this.tbUpperMin.Location =
    new System.Drawing.Point(176, 72);
this.tbUpperMin.Name = "tbUpperMin";
this.tbUpperMin.Size = new System.Drawing.Size(48, 20);
this.tbUpperMin.TabIndex = 2;
this.tbUpperMin.Text = "36";
```

```
this.tbUpperMin.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbUpperSec
//
this.tbUpperSec.Location =
    new System.Drawing.Point(296, 72);
this.tbUpperSec.Name = "tbUpperSec";
this.tbUpperSec.Size = new System.Drawing.Size(48, 20);
this.tbUpperSec.TabIndex = 3;
this.tbUpperSec.Text = "57.76";
this.tbUpperSec.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbLowerSec
//
this.tbLowerSec.Location =
    new System.Drawing.Point(296, 152);
this.tbLowerSec.Name = "tbLowerSec";
this.tbLowerSec.Size = new System.Drawing.Size(48, 20);
this.tbLowerSec.TabIndex = 6;
this.tbLowerSec.Text = "31.56";
this.tbLowerSec.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbLowerMin
//
this.tbLowerMin.Location =
    new System.Drawing.Point(176, 152);
this.tbLowerMin.Name = "tbLowerMin";
this.tbLowerMin.Size = new System.Drawing.Size(48, 20);
this.tbLowerMin.TabIndex = 5;
this.tbLowerMin.Text = "34";
this.tbLowerMin.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbLowerDeg
//
this.tbLowerDeg.Location =
    new System.Drawing.Point(56, 152);
this.tbLowerDeg.Name = "tbLowerDeg";
this.tbLowerDeg.Size = new System.Drawing.Size(48, 20);
this.tbLowerDeg.TabIndex = 4;
this.tbLowerDeg.Text = "36";
this.tbLowerDeg.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// label1
//
this.label1.Location = new System.Drawing.Point(352, 160);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(56, 16);
this.label1.TabIndex = 0;
this.label1.Text = "Seconds";
//
// label4
//
```

```
this.label4.Location = new System.Drawing.Point(232, 160);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(48, 16);
this.label4.TabIndex = 0;
this.label4.Text = "Minutes";
//
// label5
//
this.label5.Location = new System.Drawing.Point(112, 160);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(48, 16);
this.label5.TabIndex = 0;
this.label5.Text = "Degrees";
//
// label6
//
this.label6.Location = new System.Drawing.Point(24, 128);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(88, 16);
this.label6.TabIndex = 0;
this.label6.Text = "Lower Latitude:";
//
// tbLeftSec
//
this.tbLeftSec.Location =
    new System.Drawing.Point(296, 232);
this.tbLeftSec.Name = "tbLeftSec";
this.tbLeftSec.Size = new System.Drawing.Size(48, 20);
this.tbLeftSec.TabIndex = 9;
this.tbLeftSec.Text = "10.00";
this.tbLeftSec.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbLeftMin
//
this.tbLeftMin.Location =
    new System.Drawing.Point(176, 232);
this.tbLeftMin.Name = "tbLeftMin";
this.tbLeftMin.Size = new System.Drawing.Size(48, 20);
this.tbLeftMin.TabIndex = 8;
this.tbLeftMin.Text = "54";
this.tbLeftMin.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbLeftDeg
//
this.tbLeftDeg.Location =
    new System.Drawing.Point(56, 232);
this.tbLeftDeg.Name = "tbLeftDeg";
this.tbLeftDeg.Size = new System.Drawing.Size(48, 20);
this.tbLeftDeg.TabIndex = 7;
this.tbLeftDeg.Text = "121";
this.tbLeftDeg.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// label7
//
```

```
this.label7.Location =
    new System.Drawing.Point(352, 240);
this.label7.Name = "label7";
this.label7.Size = new System.Drawing.Size(56, 16);
this.label7.TabIndex = 0;
this.label7.Text = "Seconds";
//
// label8
//
this.label8.Location = new System.Drawing.Point(232, 240);
this.label8.Name = "label8";
this.label8.Size = new System.Drawing.Size(48, 16);
this.label8.TabIndex = 0;
this.label8.Text = "Minutes";
//
// label9
//
this.label9.Location = new System.Drawing.Point(112, 240);
this.label9.Name = "label9";
this.label9.Size = new System.Drawing.Size(48, 16);
this.label9.TabIndex = 0;
this.label9.Text = "Degrees";
//
// label10
//
this.label10.Location = new System.Drawing.Point(24, 208);
this.label10.Name = "label10";
this.label10.Size = new System.Drawing.Size(80, 16);
this.label10.TabIndex = 0;
this.label10.Text = "Left Longitude:";
//
// tbRightSec
//
this.tbRightSec.Location =
    new System.Drawing.Point(296, 312);
this.tbRightSec.Name = "tbRightSec";
this.tbRightSec.Size = new System.Drawing.Size(48, 20);
this.tbRightSec.TabIndex = 12;
this.tbRightSec.Text = "57.94";
this.tbRightSec.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbRightMin
//
this.tbRightMin.Location =
    new System.Drawing.Point(176, 312);
this.tbRightMin.Name = "tbRightMin";
this.tbRightMin.Size = new System.Drawing.Size(48, 20);
this.tbRightMin.TabIndex = 11;
this.tbRightMin.Text = "50";
this.tbRightMin.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// tbRightDeg
//
this.tbRightDeg.Location =
    new System.Drawing.Point(56, 312);
```

```
this.tbRightDeg.Name = "tbRightDeg";
this.tbRightDeg.Size = new System.Drawing.Size(48, 20);
this.tbRightDeg.TabIndex = 10;
this.tbRightDeg.Text = "121";
this.tbRightDeg.TextAlign =
    System.Windows.Forms.HorizontalAlignment.Right;
//
// label11
//
this.label11.Location = new System.Drawing.Point(352, 320);
this.label11.Name = "label11";
this.label11.Size = new System.Drawing.Size(56, 16);
this.label11.TabIndex = 0;
this.label11.Text = "Seconds";
//
// label12
//
this.label12.Location = new System.Drawing.Point(232, 320);
this.label12.Name = "label12";
this.label12.Size = new System.Drawing.Size(48, 16);
this.label12.TabIndex = 0;
this.label12.Text = "Minutes";
//
// label13
//
this.label13.Location = new System.Drawing.Point(112, 320);
this.label13.Name = "label13";
this.label13.Size = new System.Drawing.Size(48, 16);
this.label13.TabIndex = 0;
this.label13.Text = "Degrees";
//
// label14
//
this.label14.Location = new System.Drawing.Point(24, 288);
this.label14.Name = "label14";
this.label14.Size = new System.Drawing.Size(88, 16);
this.label14.TabIndex = 0;
this.label14.Text = "Right Longitude:";
//
// button1
//
this.button1.Location = new System.Drawing.Point(240, 384);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(72, 24);
this.button1.TabIndex = 13;
this.button1.Text = "OK";
this.button1.Click +=
    new System.EventHandler(this.button1_Click);
//
// button2
//
this.button2.Location = new System.Drawing.Point(328, 384);
this.button2.Name = "button2";
this.button2.Size = new System.Drawing.Size(72, 24);
this.button2.TabIndex = 14;
this.button2.Text = "Clear All";
this.button2.Click +=
```

```
            new System.EventHandler(this.button2_Click);
        //
        // FormNewMapProp
        //
        this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
        this.ClientSize = new System.Drawing.Size(416, 422);
        this.Controls.Add(this.button2);
        this.Controls.Add(this.button1);
        this.Controls.Add(this.tbRightSec);
        this.Controls.Add(this.tbRightMin);
        this.Controls.Add(this.tbRightDeg);
        this.Controls.Add(this.label11);
        this.Controls.Add(this.label12);
        this.Controls.Add(this.label13);
        this.Controls.Add(this.label14);
        this.Controls.Add(this.tbLeftSec);
        this.Controls.Add(this.tbLeftMin);
        this.Controls.Add(this.tbLeftDeg);
        this.Controls.Add(this.label7);
        this.Controls.Add(this.label8);
        this.Controls.Add(this.label9);
        this.Controls.Add(this.label10);
        this.Controls.Add(this.tbLowerSec);
        this.Controls.Add(this.tbLowerMin);
        this.Controls.Add(this.tbLowerDeg);
        this.Controls.Add(this.label1);
        this.Controls.Add(this.label4);
        this.Controls.Add(this.label5);
        this.Controls.Add(this.label6);
        this.Controls.Add(this.tbUpperSec);
        this.Controls.Add(this.tbUpperMin);
        this.Controls.Add(this.tbUpperDeg);
        this.Controls.Add(this.label3);
        this.Controls.Add(this.label2);
        this.Controls.Add(this.lblUpperDeg);
        this.Controls.Add(this.lblUpperLat);
        this.Controls.Add(this.lblInst);
        this.Name = "FormNewMapProp";
        this.StartPosition =
            System.Windows.Forms.FormStartPosition.CenterScreen;
        this.Text = "New Map Properties";
        this.ResumeLayout(false);

}
#endregion
/// <summary>
/// The "Clear Form" button event handler.  Resets all
/// values to zero.
/// </summary>
private void button2_Click(object sender, System.EventArgs e)
{
    this.tbUpperDeg.Text = "00";
    this.tbUpperMin.Text = "00";
    this.tbUpperSec.Text = "00.0";
    this.tbLowerSec.Text = "00.0";
    this.tbLowerMin.Text = "00";
    this.tbLowerDeg.Text = "00";
```

```
            this.tbLeftSec.Text = "00.0";
            this.tbLeftMin.Text = "00";
            this.tbLeftDeg.Text = "000";
            this.tbRightSec.Text = "00.0";
            this.tbRightMin.Text = "00";
            this.tbRightDeg.Text = "000";
        }

        /// <summary>
        /// The "OK" button event handler.  Writes all  values
        /// entered to MapHandler and closes the form.
        /// </summary>
        private void button1_Click(object sender, System.EventArgs e)
        {
            myMap.upperDeg = this.tbUpperDeg.Text;
            myMap.upperMin = this.tbUpperMin.Text;
            myMap.upperSec = this.tbUpperSec.Text;
            myMap.lowerSec = this.tbLowerSec.Text;
            myMap.lowerMin = this.tbLowerMin.Text;
            myMap.lowerDeg = this.tbLowerDeg.Text;
            myMap.leftSec = this.tbLeftSec.Text;
            myMap.leftMin = this.tbLeftMin.Text;
            myMap.leftDeg = this.tbLeftDeg.Text;
            myMap.rightSec = this.tbRightSec.Text;
            myMap.rightMin = this.tbRightMin.Text;
            myMap.rightDeg = this.tbRightDeg.Text;

            this.Close();
        }
    }
}
```

4. DeNSServer.cs

```
using System;
using System.Collections;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.IO;

namespace DeNS
{
    /// <summary>
    /// Description:
    /// The DeNSServer handles all server specific functionality
    /// involved with managing the TCP/IP connection and runs as a
    /// seperate thread.  It also manages data received in terms
    /// of storage and retreival.
    ///
    /// Requirements:
    /// an instance of PlotTimer to be passed to it's constructor.
    /// </summary>

    // Typical NMEA Sentence:
    //
```

```
// $GPGGA,123519,4807.038,N,01131.000,E, (cont on next line)
//          1,08,0.9,545.4,M,46.9,M,,*47
//  Where:
//      GGA          Global Positioning System Fix Data
//      123519       Fix taken at 12:35:19 UTC
//      4807.038,N   Latitude 48 deg 07.038' N
//      01131.000,E  Longitude 11 deg 31.000' E
//      1            Fix quality: 0 = invalid
//                1 = GPS fix (SPS)
//                2 = DGPS fix
//                3 = PPS fix
//                4 = Real Time Kinematic
//                5 = Float RTK
//                6 = estimated (dead reckoning) (2.3 feature)
//                7 = Manual input mode
//                8 = Simulation mode
//      08           Number of satellites being tracked
//      0.9          Horizontal dilution of position
//      545.4,M      Altitude, Meters, above mean sea level
//      46.9,M       Height of geoid (mean sea level) above WGS84
//                   ellipsoid
//      (empty field) time in seconds since last DGPS update
//      (empty field) DGPS station ID number
//      *47          the checksum data, always begins with *

public class DeNSServer
{

    private ArrayList posArrList;
    private ArrayList msgArrList;
    private int lastDrawn;
    private int lastMsg;
    private Thread readThread;
    private Socket conn;
    private NetworkStream stream;
    private BinaryWriter writer;
    private BinaryReader reader;
    private System.Windows.Forms.Timer myPlotTimer;
    public bool positGreen;

    /// <summary>
    /// Class constructor.  Requires an instance of plotTimer.
    /// </summary>
    public DeNSServer(System.Windows.Forms.Timer plotTimer)
    {
        //start thread that runs server.
        readThread = new Thread(new ThreadStart(runServer));
        readThread.Start();

        //Initialize data strux and indexes for use storing
        //  positions and messages
        posArrList = new ArrayList(1);
        lastDrawn = -1;
        msgArrList = new ArrayList(1);
        lastMsg = -1;

        //Set default position to indicate within track
```

```
    //  toloerance
    positGreen = true;

    //instantiate a new plotTimer to point to the one passed.
    myPlotTimer = plotTimer;
}

/// <summary>
/// Used in early stages of development to load dummy posits
/// into posArrList for testing and verification purposes.
/// </summary>
public void testPosDraw()
{
    string s1 = "$GPGGA,120000,3636.834,N,12154.100,W,1,08" +
                ",0.9,545.4,M,46.9,M,,*47";
    string s2 = "$GPGGA,120100,3636.750,N,12154.100,W,1,08" +
                ",0.9,545.4,M,46.9,M,,*47";
    string s3 = "$GPGGA,120200,3636.750,N,12153.900,W,1,08" +
                ",0.9,545.4,M,46.9,M,,*47";

    addNewPos(s1);
    addNewPos(s2);
    addNewPos(s3);
}

/// <summary>
/// The main thread that manages the server side of
/// communications and handles incoming messages in
/// the appropriate manner.
/// </summary>
private void runServer()
{
    TcpListener listener;
    int cnt = 1;
    string newPos = "";

    try
    {
        //Create a listener on all assigned IP addresses
        //  using port 5000
        listener = new TcpListener(IPAddress.Any, 5000);
        listener.Start();

        while (true)
        {
            //Display server status to user
            MessageBox.Show("Waiting for Connection...");

            //Establish TCP/IP connection and resources
            conn = listener.AcceptSocket();
            stream = new NetworkStream(conn);
            writer = new BinaryWriter(stream);
            reader = new BinaryReader(stream);

            //Display server status to user
            MessageBox.Show("Connection " + cnt +
                            " received...");
```

```
//Send connection status to MobileDeNS user
writer.Write("Connection Successful...");

//Initialize reply string
string reply = "";

//Process incoming data while connected
do
{
    try
    {
        //Read a packet in
        reply = reader.ReadString();

        //Packet is a GPS position
        if(reply.Substring(0,1) == "G")
        {
            //strip the packet type header and
            //  add it to posArrList
            newPos = reply.Substring(1,
                    (reply.Length - 1));
            addNewPos(newPos);
        }
        //Packet is a message from MobileDeNS
        else if(reply.Substring(0,1) == "M")
        {
            //strip the packet type header and
            //  add it to msgArrList
            addNewMsg(reply.Substring(1,
                    (reply.Length - 1)));
        }
        //Packet indicates that last position is
        //  out of track tolerence
        else if(reply.Substring(0,1) == "X")
        {
            //set cursor color flag to alert user
            positGreen = false;
        }
        //Packet indicates that last position is
        //  within track tolerence
        else if(reply.Substring(0,1) == "W")
        {
            //set cursor color flag to alert user
            positGreen = true;
        }
        else
        //Packet received was not a known type
        {
            MessageBox.Show("unrecognized " +
                            "packet received.");
        }
    }
    //Catch and display error type
    catch (Exception error)
    {
        MessageBox.Show("Error in runServer do " +
```

```
                                "loop: " + error.ToString());
                }
            }while(conn.Connected);

            //Increment connection counter
            cnt++;

            //Close the connection
            closeConn();
        }
    }
    //Catch and display error type
    catch (Exception error)
    {
        MessageBox.Show("Error in runServer listener loop: " +
                        error.ToString());
    }
}

/// <summary>
/// Closes TCP/IP resources and connection.
/// </summary>
private void closeConn()
{
    //Close TCP/IP resourses and connection
    writer.Close();
    reader.Close();
    stream.Close();
    conn.Close();

    //Display connection status to user
    MessageBox.Show("Connection closed.");
}

/// <summary>
/// Determines whether a posit exists in posArrList that has
/// not yet been drawn.
/// </summary>
public bool newPosExists()
{
    //lastDrawn var + 1 indicates a position index that
    //  has not yet been displayed.  posArrList.Count is
    //  the number of elements contained in the Array List
    if(posArrList.Count == (lastDrawn + 1))
    {
        //New position does not exist
        return false;
    }
    else
    {
        //New position does exist
        return true;
    }
}

/// <summary>
/// Determines whether a message exists in msgArrList that has
```

```
/// not yet been displayed.
/// </summary>
public bool newMsgExists()
{
    //lastMsg var + 1 indicates a message index that
    //  has not yet been displayed.  msgArrList.Count is
    //  the number of elements contained in the Array List
    if(msgArrList.Count == (lastMsg + 1))
    {
        //New position does not exist
        return false;
    }
    else
    {
        //New position does exist
        return true;
    }
}

/// <summary>
/// Retreives a new position and increments the lastDrawn var
/// that tracks the index of the last posit displayed.
/// </summary>
public string getNewPos()
{
    string result;

    //Convert oldest element of posArrList not yet drawn to a
    //  string
    result = posArrList[++lastDrawn].ToString();

    //return the result of above operation
    return result;
}

/// <summary>
/// Retreives a new message and increments the lastMsg var
/// that tracks the index of the last message displayed.
/// </summary>
public string getNewMsg()
{
    string result;

    //Convert oldest element of msgArrList not yet displayed
    //  to a string
    result = msgArrList[++lastMsg].ToString();

    //return the result of above operation
    return result;
}

/// <summary>
/// Adds a new position to posArrList.
/// </summary>
public void addNewPos(string newPos)
{
    //add newPos argument to posArrList
```

```
            posArrList.Add(newPos);
        }

        /// <summary>
        /// Adds a new message to msgArrList.
        /// </summary>
        public void addNewMsg(string newMsg)
        {
            //add newMsg argument to msgArrList
            msgArrList.Add(newMsg);
        }

        /// <summary>
        /// initializes posArrList and resets the lastDrawn var
        /// that tracks the index of the last posit displayed.
        /// </summary>
        public void initPosArrList()
        {
            //Clear posArrList of all previous positions and reset
            //  lastDrawn index
            posArrList.Clear();
            lastDrawn = -1;
        }
    }
}
```

5. MapHandler.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace DeNS
{
    /// <summary>
    /// MapHandler Class stores all currently loaded map parameters
    /// and implements hardcoded lat and long per pixel values.  It
    /// also provides the method which allows the user to browse
    /// to a map and retreives that maps name, height, width, and
    /// path.
    /// </summary>
    public class MapHandler
    {
        private string mapName;
        private int mapHeight;
        private int mapWidth;
        public string upperDeg;
        public string upperMin;
        public string upperSec;
        public string lowerDeg;
        public string lowerMin;
        public string lowerSec;
        public string leftDeg;
        public string leftMin;
        public string leftSec;
        public string rightDeg;
        public string rightMin;
```

```
        public string rightSec;
        public float latPerPixel;
        public float longPerPixel;

        /// <summary>
        /// MapHandler constructor.
        /// </summary>
        public MapHandler()
        {
            //Set lat and pong per pixel values for Monterey Bay
            //  test jpg
            latPerPixel = .1606813F;
            longPerPixel = .1607197F;
        }
        /// <summary>
        /// Provides OpenFileDialog box that allows user to pick
        /// a map.  returns map file as bitmap.
        /// </summary>
        public Bitmap getMap()
        {
            //retreives the specified map file
            OpenFileDialog openFile = new OpenFileDialog();
            Bitmap newMap = new Bitmap(896, 577);

            //if map was chosen
            if(openFile.ShowDialog() == DialogResult.OK)
            {
                //get map file
                newMap = new Bitmap(openFile.FileName);
                //get map name
                mapName = openFile.FileName;
                //get map height
                mapHeight = newMap.Height;
                //get map width
                mapWidth = newMap.Width;
            }

            //return map file as bitmap
            return newMap;
        }
    }
}
```

6. PktHandler.cs

```
using System;
using System.Windows.Forms;

namespace DeNS
{
    /// <summary>
    /// PktHandler Class allows storage, manipulation, and retreival
    /// of pre-processed GPS data recieved from MobileDeNS.
    /// </summary>
    public class PktHandler
    {
        private string packetType;
```

```
private string fixTime;
private float latNum;
private int latDeg;
private int latMin;
private float latSec;
private string latHem;
private float longNum;
private int longDeg;
private int longMin;
private float longSec;
private string longHem;
private string satNum;
private int plotX;
private int plotY;

/// <summary>
/// Default Plotter constructor, requires no arguments and
/// creates an empty instance of PktHandler
/// </summary>
public PktHandler()
{

}

/// <summary>
/// PktHandler constructor, requires a string to be passed
/// as an argument, which it parses as required.
/// </summary>
public PktHandler(string packet)
{
    packetType = packet.Substring(0, 3);
    latNum = float.Parse(packet.Substring(3,8));
    latDeg = int.Parse(packet.Substring(3,2));
    latMin = int.Parse(packet.Substring(5,2));
    latSec = float.Parse(packet.Substring(7,4)) * 60;
    latHem = packet.Substring(11,1);
    longNum = float.Parse(packet.Substring(12,9));
    longDeg = int.Parse(packet.Substring(12,3));
    longMin = int.Parse(packet.Substring(15,2));
    longSec = float.Parse(packet.Substring(17,4)) * 60;
    longHem = packet.Substring(21,1);
    satNum = packet.Substring(22,2);
}

/// <summary>
/// PktHandler constructor, requires two strings to be passed
/// as arguments, one of which is the fix time, and the
/// other of which it parses as required.
/// </summary>
public PktHandler(string time, string data)
{
    fixTime = time;
    packetType = data.Substring(0,3);
    latNum = float.Parse(data.Substring(3,8));
    latDeg = int.Parse(data.Substring(3,2));
    latMin = int.Parse(data.Substring(5,2));
    latSec = float.Parse(data.Substring(7,4)) * 60;
```

```
        latHem = data.Substring(11,1);
        longNum = float.Parse(data.Substring(12,9));
        longDeg = int.Parse(data.Substring(12,3));
        longMin = int.Parse(data.Substring(15,2));
        longSec = float.Parse(data.Substring(17,4)) * 60;
        longHem = data.Substring(22,1);
        satNum = data.Substring(23,2);
    }
    /// <summary>
    /// Allows the x and y pixel coordinates of the position to
    /// be set
    /// </summary>
    public void setPlot(int x, int y)
    {
        plotX = x;
        plotY = y;
    }
    /// <summary>
    /// Returns the x pixel coordinate.
    /// </summary>
    public int getPlotX()
    {
        return plotX;
    }
    /// <summary>
    /// Returns the y pixel coordinate.
    /// </summary>
    public int getPlotY()
    {
        return plotY;
    }
    /// <summary>
    /// Returns the latitude number as a float in Decimal Degrees.
    /// </summary>
    public float getLatNum()
    {
        return latNum;
    }
    /// <summary>
    /// Returns the longitude number as a float in Decimal
    /// Degrees.
    /// </summary>
    public float getLongNum()
    {
        return longNum;
    }
    /// <summary>
    /// Return the latitude as a string seperated by the words
    /// "degrees", "minutes", and "seconds", along with the
    /// one letter hemipshere indicator.
    /// </summary>
    public string getLat()
    {
        string lat;

        lat = latDeg.ToString() + " degrees ";
        lat = lat + latMin.ToString() + " Minutes ";
```

```
        lat = lat + latSec.ToString() + " Seconds ";
        lat = lat + " " + latHem;

        return lat;
    }
    /// <summary>
    /// Return the longitude as a string seperated by the words
    /// "degrees", "minutes", and "seconds", along with the
    /// one letter hemipshere indicator.
    /// </summary>
    public string getLong()
    {
        string lon;

        lon = longDeg.ToString() + " degrees ";
        lon = lon + longMin.ToString() + " Minutes ";
        lon = lon + longSec.ToString() + " Seconds ";
        lon = lon + longHem;

        return lon;
    }
    /// <summary>
    /// Allows the packetType to be set to a string passed as a
    /// parameter.
    /// </summary>
    public void setType(string type)
    {
        packetType = type;
    }
    /// <summary>
    /// Returns the packetType as a string.
    /// </summary>
    public string getType()
    {
        return packetType;
    }
    /// <summary>
    /// Allows the fixTime to be set to a string passed as a
    /// parameter in the form "HH:MM:SS".
    /// </summary>
    public void setTime(string time)
    {
        fixTime = time.Substring(0,2) + ":" +
            time.Substring(2,2) + ":" +
            time.Substring(4,2);
    }
    /// <summary>
    /// Returns the fixTime as a string.
    /// </summary>
    public string getTime()
    {
        return fixTime;
    }
    /// <summary>
    /// Allows the latitude to be set to a float passed as a
    /// parameter in decimal degrees and the hemisphere to be set
    /// to a string representing the one letter hemisphere
```

```
/// designator.
/// </summary>
public void setLat(float lat, string hem)
{
    latNum = lat;
    latHem = hem;
}
/// <summary>
/// Returns the seconds of latitude as a float.
/// </summary>
public float getLatSecs()
{
    return latSec;
}
/// <summary>
/// Returns the minutes of latitude as a float.
/// </summary>
public float getLatMins()
{
    return latMin;
}
/// <summary>
/// Allows the longitude to be set to a float passed as a
/// parameter in decimal degrees and the hemisphere to be set
/// to a string representing the one letter hemisphere
/// designator.
/// </summary>
public void setLong(float lon, string hem)
{
    longNum = lon;
    longHem = hem;
}
/// <summary>
/// Returns the seconds of longitude as a float.
/// </summary>
public float getLongSecs()
{
    return longSec;
}
/// <summary>
/// Returns the minutes of longitude as a float.
/// </summary>
public float getLongMins()
{
    return longMin;
}
/// <summary>
/// Allows the number of satellites to be set to a string
/// passed as a parameter.
/// </summary>
public void setSatNum(string sat)
{
    satNum = sat;
}
/// <summary>
/// Returns the number of satellites as a string.
/// </summary>
```

```
        public string getSatNum()
        {
            return satNum;
        }
    }
}
```

7. Plotter.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace DeNS
{
    /// <summary>
    /// Plotter Class handles calculations which take place to
    /// determine screen position of positional cursor for display
    /// to the user.
    /// </summary>
    public class Plotter
    {
        private DeNSServer myServer;
        private MapHandler myMapper;
        private PktHandler pkt;

        /// <summary>
        /// Plotter constructor, requires an instance of DeNSServer,
        /// and MapHandler to be passed as arguments
        /// </summary>
        public Plotter(DeNSServer server, MapHandler mapper)
        {
            //set pointers to arguments
            myServer = server;
            myMapper = new MapHandler();
            myMapper = mapper;
        }

        /// <summary>
        /// Calculates the x and y pixel positions at which the
        /// positional cursor should be placed.  Returns a Point
        /// whose x and y coordinates represent the location of
        /// the positional cursor location.
        /// </summary>
        public Point calcScrPos()
        {
            Point scrPos = new Point();

            //create a new instance of PktHandler
            pkt = new PktHandler(myServer.getNewPos());

            //set the x and y coordinates of the new packet
            pkt.setPlot(calcX(pkt.getLongSecs(), pkt.getLongMins()),
                        calcY(pkt.getLatSecs(), pkt.getLatMins()));

            //retreive the x and y coordinates of the new packet
            scrPos.X = pkt.getPlotX();
```

```
    scrPos.Y = pkt.getPlotY();

    //return the x and y coordinates as a point
    return scrPos;
}

/// <summary>
/// Receives longitudinal minutes and seconds in float
/// format and converts them into the x pixel coordinate
/// of the positional cursor which is returned as an integer.
/// </summary>
public int calcX(float lonSecs, float lonMins)
{
    float longDif;
    float leftMinSec;
    float tempDif;
    int x;

    //convert the left longtitude minutes and seconds
    //  parameter to a combined decimal float
    leftMinSec = float.Parse(myMapper.leftMin) +
                 (float.Parse(myMapper.leftSec) / 60);
    tempDif = (leftMinSec % 1) * 60;
    tempDif = tempDif + ((leftMinSec - (tempDif / 60)) * 60);
    //subtract combined arguments for long  minutes and
    //  seconds from above result to get the distance from
    //  the left-most edge of the map
    longDif = tempDif - ((lonMins * 60) + lonSecs);
    //cast the above float to an integer to get the x pixel
    //  coordinate
    x = (int)(longDif / myMapper.longPerPixel);

    //return the x pixel coordinate
    return x;
}

/// <summary>
/// Receives latitudinal minutes and seconds in float
/// format and converts them into the y pixel coordinate
/// of the positional cursor which is returned as an integer.
/// </summary>
public int calcY(float latSecs, float latMins)
{
    float latDif;
    float upperMinSec;
    float tempDif;
    int y;

    //convert the upper latitude minutes and seconds
    //  parameter to a combined decimal float
    upperMinSec = float.Parse(myMapper.upperMin) +
                  (float.Parse(myMapper.upperSec) / 60);
    tempDif = (upperMinSec % 1) * 60;
    tempDif = tempDif + ((upperMinSec - (tempDif /60)) * 60);
    //subtract combined arguments for lat  minutes and
    //  seconds from above result to get the distance from
    //  the upper-most edge of the map
```

```
            latDif = tempDif - ((latMins * 60) + latSecs);
            //cast the above float to an integer to get the y pixel
            //  coordinate
            y = (int)(latDif / myMapper.latPerPixel);

            //return the y pixel coordinate
            return y;
        }
    }
}
```

THIS PAGE LEFT INTENTIONALLY BLANK

LIST OF REFERENCES

Albahari, Ben, Peter Drayton, and Brad Merrill. C# Essentials. USA: O'Reilly, 2001.

Airespace Technology. "Introduction to WiMax." Airespace.com. 2005. Last accessed September 4, 2006 <www.airespace.com/technology/technote_introduction_to_wimax.php>

Antonello, G., H. Yaghoobi, and A Agrawal. "Wimax Technical Working Group." Wimax Forum. January 20, 2004. Last accessed September 4, 2006 <www.wimaxforum.org/news/events/wimax_day_agenda/Gordon_Member_IEEE_802.16.pdf>

Deitel, H. M., P. J. Deitel, J. A. Listfield, T. R. Nieto, C. H. Yaeger, and M. Zlatkina. C# for Experienced Programmers. New Jersey: Prentice Hall, 2003.

Earhart, Eugene. Email. June 21,2005.

ESRI. ArcIMS 9: Getting Started with ArcIMS. USA: ESRI Press, 2004.

———. ArcGIS 9: Getting Started with ArcGIS. USA: ESRI Press, 2005.

ESRI GIS and Mapping Software. August 31, 2006. ESRI. August 31, 2006. Last accessed September 4, 2006 <http://www.esri.com>

Foxall, James. Microsoft Visual C# . NET 2003. Indiana: Sams Publishing, 2004.

Gombert, Gregory. Email. June 18, 2005.

Gunnerson, Eric. A Programmer's Introduction to C#. New York: Apress, 2000.

Highpoint Software. GPSAccess Bluetooth GPS Library Developer's Guide v1.0. Highpoint Software, 2005.

Hoffman-Wellenhof, B., H. Lichtenegger, and J. Collins. Global Positioning System: Theory and Practice. 5th Ed. Springer, September 23, 20004.

Institute of Electrical and Electronics Engineering (IEEE), Inc. Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Wireless Personal Area Networks (WPANs). New York. IEEE: 2005.

Jaragin, J. GPS/GIS in a Windows CE Environment. Proceedings of the 22nd Annual ESRI User Conference, Houston, Texas: 2002

Mettala, Riku. Bluetooth Protocol Architecture. August 25, 1999.

Singh, Gurminder. Lecture notes. Device Independence - II. 2005. IEEE. January 15, 2006. Last accessed September 4, 2006 <ieeexplore.ieee.org /iel5/93/28183/01261102.pdf?isnumber=28183&arnumber=1261102>

———. “Content Repurposing.” IEEE Proceedings Vol I, 2004. IEEE. January 15, 2006. Last accessed September 4, 2006<ieeexplore.ieee.org /iel5/93/28183/01261102.pdf?isnumber=28183&arnumber=1261102>

USA. Department of Defense. Interface Standard for Vector Product Format. United States: Department of Defense, 1996.

———. Department of Defense. Performance Specification: Digital Nautical Chart. United States: Department of Defense, 1997.

USN. Office of the Chief of Naval Operations. Navigator of the Navy Geospatial Information and Navigation 001. US Navy: Chief of Naval Operations, DTG 171228Z JUL 01.

———. Office of the Chief of Naval Operations. Navigator of the Navy Geospatial Information and Navigation 002. US Navy: Chief of Naval Operations, DTG 010030Z FEB 02.

———. Office of the Chief of Naval Operations. Navigator of the Navy Geospatial Information and Navigation (GIN) 003. US Navy: Chief of Naval Operations, DTG 011600Z AUG 02.

———. Office of the Chief of Naval Operations. Navigator of the Navy Geospatial Information and Navigation (GIN) 004. US Navy: Chief of Naval Operations, DTG 031325Z JUN 03.

Wigley, Andy, and Peter Roxburgh. Building .NET Applications for Mobile Devices. Washington: Microsoft Press, 2002.

Yao, Paul, and David Durant. .NET Compact Framework Programming with C#. Massachusetts: Addison Wesley, 2004.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California